EXHIBIT 4

U.S. Patent No. 6,813,742 – ASUSTeK Computer Inc.

Claims 1 and 6

TurboCode LLC ("TurboCode") provides evidence of infringement of claim 6 of U.S. Patent No. 6,813,742 (hereinafter "the '742 patent") by ASUSTeK Computer Inc ("Asus" or "Defendant"). In support thereof, TurboCode provides the following claim charts.

"Accused Instrumentalities" as used herein refers to at least cellular telephones, tablet computers, and/or other devices with 3G and/or 4G/LTE capabilities and that comply with the 3G and/or 4G/LTE standards as disclosed in the 3rd Generation Partnership Project ("3GPP") Standard Specifications governing cellular wireless communications (e.g., TS 26.071-26.999), and similar systems, products, and/or devices (including, but not limited to the Asus ZenFone 7 Pro). These claim charts demonstrate Asus's infringement, and provide notice of such infringement, by comparing each element of the asserted claims to corresponding components, aspects, and/or features of the Accused Instrumentalities. These claim charts are not intended to constitute an expert report on infringement. These claim charts include information provided by way of example, and not by way of limitation.

The analysis set forth below is based only upon information from publicly available resources regarding the Accused Instrumentalities, as Asus has not yet provided any non-public information. An analysis of Asus's (or other third parties') technical documentation and/or software source code may assist in fully identify all infringing features and functionality. Accordingly, TurboCode reserves the right to supplement this infringement analysis once such information is made available to TurboCode. Furthermore, TurboCode reserves the right to revise this infringement analysis, as appropriate, upon issuance of a court order construing any terms recited in the asserted claims.

Unless otherwise noted, TurboCode contends that Asus directly infringes the '742 patent in violation of 35 U.S.C. § 271(a) by selling, offering to sell, making, using, and/or importing the Accused Instrumentalities. The following exemplary analysis demonstrates that infringement. Unless otherwise noted, TurboCode further contends that the evidence below supports a finding of indirect infringement under 35 U.S.C. §§ 271(b) and/or (c), in conjunction with other evidence of liability under one or more of those subsections. Asus makes, uses, sells, imports, or offers for sale in the United States, or has made, used, sold, imported, or offered for sale in the past, without authority, or induces others to make, use, sell, import, or offer for sale in the United States, or has induced others to make, use, sell, import, or offer for sale in the past, without authority products, equipment, or services that infringe claims 1 and 6 of the '742 patent, including without limitation, the Accused Instrumentalities.

Unless otherwise noted, TurboCode believes and contends that each element of each claim asserted herein is literally met through Asus's provision of the Accused Instrumentalities. However, to the extent that Asus attempts to allege that any asserted claim element is not literally met, TurboCode believes and contends that such elements are met under the doctrine of equivalents. More specifically, in its investigation and analysis of the Infringing Instrumentalities, TurboCode did not identify any substantial differences between the elements of the patent claims and the corresponding features of the Accused Instrumentalities, as set forth herein. In each instance, the identified feature of the Accused Instrumentalities performs at least substantially the same function in substantially the same way to achieve substantially the same result as the corresponding claim element.

To the extent the chart of an asserted claim relies on evidence about certain specifically-identified Accused Instrumentalities, TurboCode asserts that, on information and belief, any similarly-functioning instrumentalities also infringes the charted claim. TurboCode reserves the right to amend

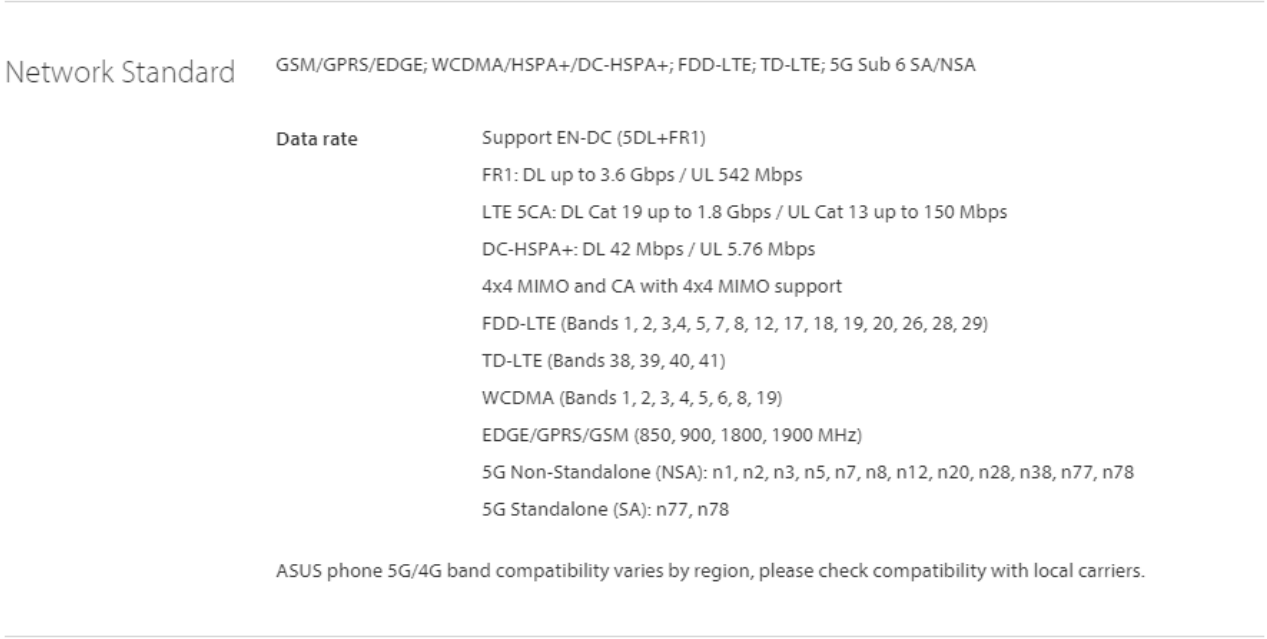
this infringement analysis based on other products made, used, sold, imported, or offered for sale by Asus. TurboCode also reserves the right to amend this infringement analysis by citing other claims of the '742 patent, not listed in the claim chart, that are infringed by the Accused Instrumentalities. TurboCode further reserves the right to amend this infringement analysis by adding, subtracting, or otherwise modifying content in the "Accused Instrumentalities" column of each chart.

CLAIM CHART
U.S. Patent No. 6,813,742

Claim 1	'742 Accused Instrumentalities
1. A baseband processor for iteratively processing a plurality sequences of received baseband digital signals, the baseband processor comprising:	To the extent the preamble is found to be a limitation, as discussed below, the '742 Accused Instrumentalities perform a method of iteratively decoding a plurality of sequences of received baseband signals. The '742 Accused Instrumentalities include products, devices, systems, and components of systems that comply with the 3G and/or 4G/LTE standards as disclosed in the 3rd Generation Partnership Project ("3GPP") Standard Specifications governing cellular wireless communications and that were or are designed, developed, tested, made, used, offered for sale, sold in the United States, imported into the United States, or that have a nexus to the United States, including for example and without limitation, the following:[1] • Fonepad, PadFone, ROG Phone, and Zenfone series smartphones, MeMO Pad, Nexus, VivoTab, and ZenPad series cellular-enabled tablets, AC1900, 4G-AC86U, 4G-N12 B1, 4G-AC53U, 4G-AC55U, and 4G-AC68U series modem routers, Asus Laptop (e.g., BR1100), Chromebook, ExpertBook, NovaGo, and Transformer series cellular-enabled laptops, and similar products, devices, systems, and components of systems that comply with the 3G and/or 4G/LTE standards as disclosed in the 3rd Generation Partnership Project ("3GPP") Standard Specifications governing cellular wireless communications (*e.g.*, TS 26.071-26.999). **Direct Infringement** Asus has directly infringed and continues to directly infringe at least Claims 1 and 6 of the '742 Patent under 35 U.S.C. § 271(a) by making, using, offering to sell, selling, within the United States, or importing into the United States without authorization the '742 Accused Instrumentalities. This direct infringement is described hereinbelow. **Indirect Infringement** Asus has induced and continues to induce infringement by others of at least Claims 1 and 6 of the '742

[1] The '742 Accused Instrumentalities are identified based on information currently available on Asus's website and other public information. TurboCode reserves the right to supplement, revise or otherwise amend this list, when additional model numbers are identified and new information becomes available during discovery.

	Patent under 35 U.S.C. § 271(b) by encouraging its customers (including but not limited to manufacturers of the '742 Accused Instrumentalities or manufacturers using Asus's product designs for the '742 Accused Instrumentalities; manufacturers of computer products incorporating the '742 Accused Instrumentalities; and distributors, wholesalers, and retailers, and end users, to make, use, sell, offer to sell, and import in the United States without authorization the '742 Accused Instrumentalities of the underlying direct infringement. Asus has contributed to and continues to contribute to infringement by others of at least Claim 6 of the '742 Patent under 35 U.S.C. § 271(c) by selling, offering to sell, importing, and/or supplying in the United States without authority components of the products that infringe one or more claims of the '742 Patent, including but not limited to the '742 Accused Instrumentalities of the underlying direct infringement. Asus's ZenFone 7 Pro is a representative product because it performs iterative decoding in accordance with the 3G and/or 4G/LTE standards disclosed in the 3GPP Standard Specifications as do the '742 Accused Instrumentalities. For example, Asus's ZenFone 7 Pro complies with 3G and 4G/LTE as shown below:

	 *See* Asus ZenFone 7 Pro Specification, *available at* https://www.asus.com/Mobile/Phones/All-series/ZenFone-7-Pro/techspec/. As another example, each accused instrumentality, including the ZenFone 7 Pro, provides a baseband processor for iteratively processing a plurality sequences of received baseband digital signals. For example, the turbo decoder described in the 3GPP specification includes a baseband processor for iteratively processing a plurality of sequences of received baseband digital signals: "The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies…"

See Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf.

See also Cheng et. al. "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y:

"Figure 3 shows the system architecture. The blocks in the dashed box handle the turbo decoding operations, and those outside the dashed box belong to the DVFS scheme. Turbo decoding is an iterative process with several turbo iterations. Each turbo iteration comprises two soft-in, soft-out (SISO) decoding processes using BCJR algorithm [8]with the first one performed on the input code block in the original order and the second one in an order generated by the interleaver block."

See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 14, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

Type/Constant	Dimension	Description
/* Synchronization */		
const Int16 wakeupSin500	[16]	sine waveform at 500 Hz
const Int16 wakeupCos500	[16]	cosine waveform at 500 Hz
const Int16 wakeupSin800	[10]	sine waveform at 800 Hz
const Int16 wakeupCos800	[10]	cosine waveform at 800 Hz

See also id. at 20:

```
/*=========================================================================*/
/* PSAP FUNCTION: PsapReceiver                                             */
/*-------------------------------------------------------------------------*/
/* Description: PSAP receiver function (decoding is done outside)          */
/*                                                                         */
/* In:     const ModState* ms        -> modulator struct                   */
/*         const Int16*    pcm       -> input data for demodulation        */
/* Out:    IntLLR*         softBits  <- demodulated soft bit sequence      */
/*-------------------------------------------------------------------------*/
void PsapReceiver(const ModState *ms, const Int16 *pcm, IntLLR *softBits)


/*=========================================================================*/
/* PSAP FUNCTION: SymbolDemod                                              */
/*-------------------------------------------------------------------------*/
/* Description: symbol demodulator                                         */
/*                                                                         */
/* In:     const ModState* ms        -> modulator struct                   */
/*         const Int16*    mPulse    -> received pulse train               */
/* Out:    IntLLR*         softBits  <- demodulated soft bit sequence      */
/*-------------------------------------------------------------------------*/
void SymbolDemod(const ModState *ms, const Int16 *mPulse, IntLLR *softBits)
```

- ecall-fec.c line 163
  *Bool FecDecode(const IntLLR *in, Int16 rv, Ord1 *out)*

```
/*=========================================================================*/
/* DECODER FUNCTION: FecDecode                                             */
/*-------------------------------------------------------------------------*/
/* Description: decoding to find the MSD                                   */
/*                                                                         */
/* In:     const IntLLR* in   -> received soft bits                        */
/*         Int16         rv   -> redundancy version                        */
/* Out:    Ord1*         out  <- decoded MSD in binary representation      */
/* Return: Bool               <- result of CRC check                       */
/*-------------------------------------------------------------------------*/
Bool FecDecode(const IntLLR *in, Int16 rv, Ord1 *out)
```

	• ecall-fec.c line 240 */*iterative decoding*/* *for (i = 0; i < FEC_ITERATIONS; i++)* *See also* May et al., "A 150Mbit/s 3GPP LTE Turbo code decoder," 2010 Design, Automation & Test in Europe Conference & Exhibition (March 8-12, 2010), *available at* https://ieeexplore.ieee.org/document/5457035/authors#authors: "3GPP long term evolution (LTE) enhances the wireless communication standards UMTS and HSDPA towards higher throughput. A throughput of 150 Mbit/s is specified for LTE using 2×2 MIMO. For this, highly punctured Turbo codes with rates up to 0.95 are used for channel coding, which is a big challenge for decoder design. This paper investigates efficient decoder architectures for highly punctured LTE Turbo codes. We present a 150 Mbit/s 3GPP LTE Turbo code decoder, which is part of an industrial SDR multi-standard baseband processor chip."
an input buffer comprising at least three shift registers, for receiving an input signal and generating first, second, and third shifted input signals;	The '742 Accused Instrumentalities provide an input buffer comprising at least three shift registers, for receiving an input signal and generating first, second, and third shifted input signals. For example, the representative product ZenFone 7 Pro includes an input buffer comprising at least three shift registers. The input buffer receives an input signal, and first, second, and third shifted input signals are generated for input to a turbo decoder. The generated first, second, and third shifted input signals, shown as "soft data," "soft parity 1, ptail 1," and "soft parity 2, ptail 2," are input into the 1st constituent decoder and 2nd constituent decoder as shown below:

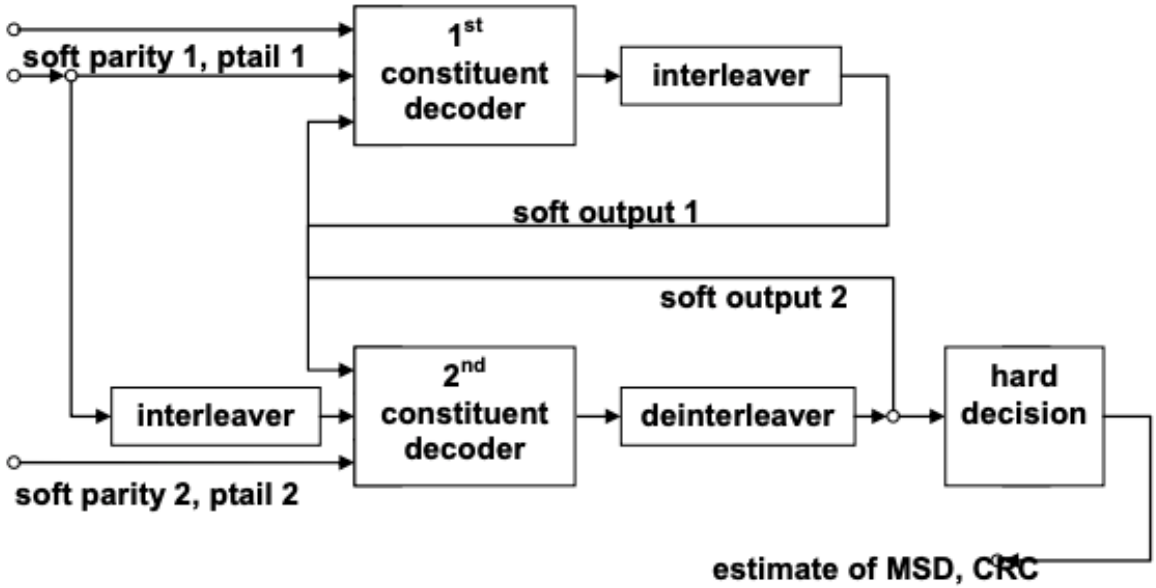


Figure 18: Turbo decoder

See "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); eCall data transfer; In-band modem solution; General description (3GPP TS 26.267 version 11.0.0 Release 11)," at 25, *available at* https://www.etsi.org/deliver/etsi_TS/126200_126299/126267/11.00.00_60/ts_126267v110000p.pdf ("3GPP TS 26.267").

Each such buffer provides three sections based on operations of a turbo encoder. For example, an input buffer is denoted as Figure 7, labeled as a "channel coded bit buffer":

Figure 7: Channel coded bit buffer

Note that the "soft data" of Figure 18 corresponds to the "MSD+CRC," "tail 1," and "tail 2" fields of Figure 7.

See 3GPP TS 26.267, at 14.

Such a channel coded bit buffer can then be decoded using shifting to generate the first, second, and third shifted input signals, which are stored in registers for input into the turbo decoder:

"6.1.3 Modulation

The encoded binary data stream bits b_i are grouped into symbols. Each symbol d_j carries 4 bits of information and modulates one basic downlink waveform.
…
Table 4 describes the symbol modulation mapping between symbol and the downlink waveform. The downlink waveform is derived from the basic downlink waveform $p_{DL}(n)$ by a cyclic right-shift by k samples, denoted by (p → k) , and multiplication with a sign q."

See 3GPP TS 26.267, at 22.

See also Cheng et. al. "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at Fig. 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y :

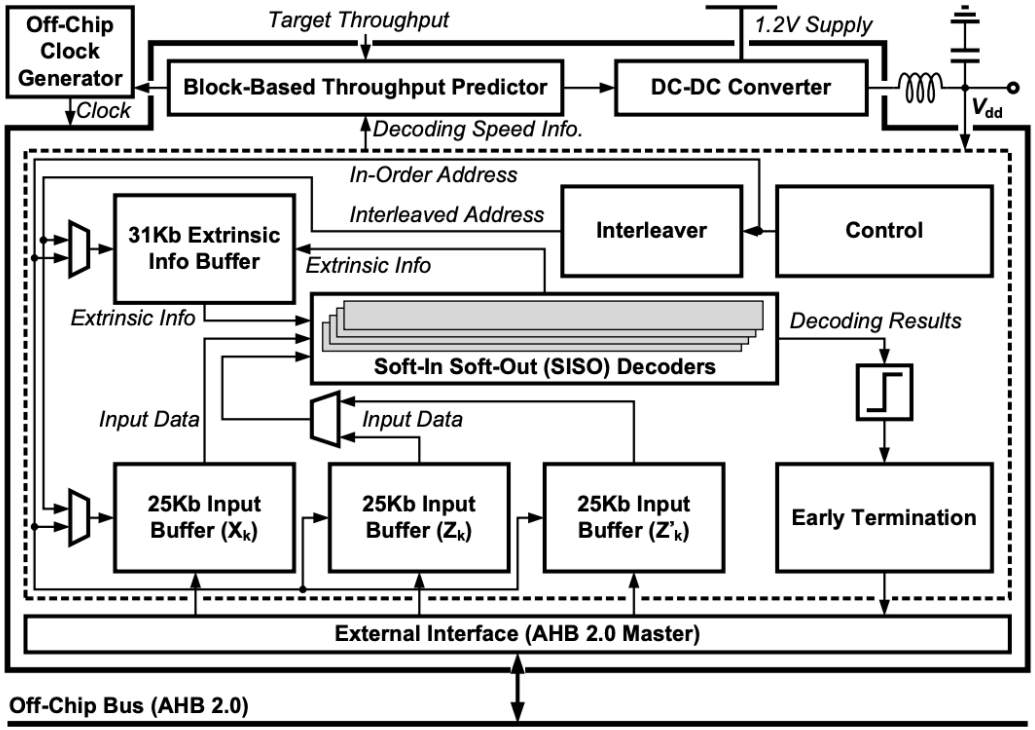
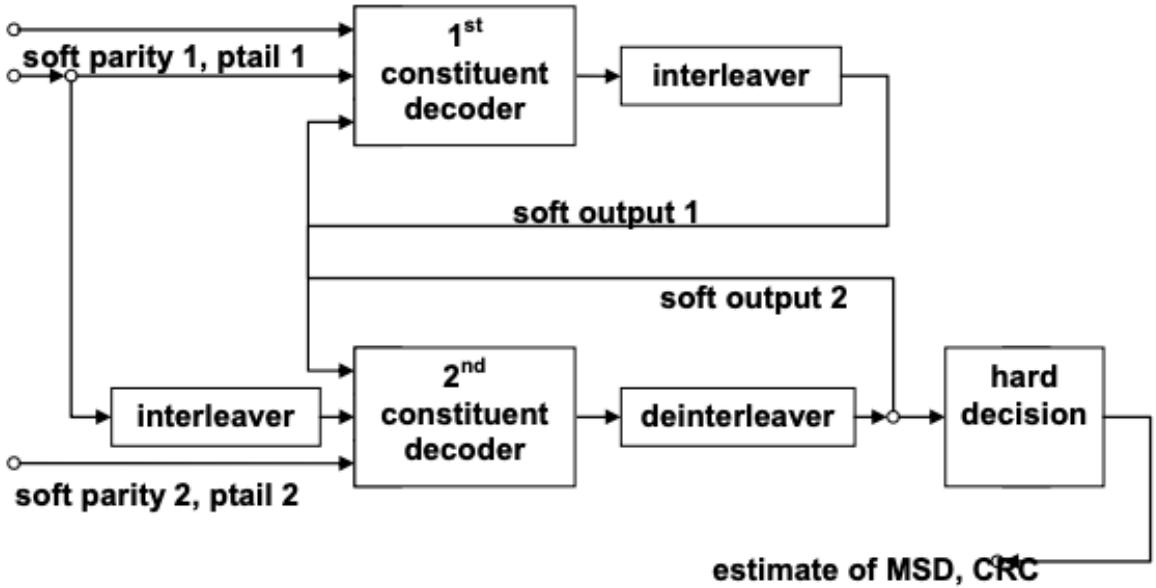


Fig. 3. The system architecture.

See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

- ecall-fec.c line 200
  *void UpdateBuffer(IntLLR *chLLRbuffer, const IntLLR *softInBits, Int16 rv)*

- ecall-fec.c line 225

	*void DecodeBuffer(const IntLLR *syst1, const IntLLR *syst2, const IntLLR *parity1, const IntLLR *parity2, Ord1 *decBits)*
at least two soft decision decoders including first and second soft decision decoders serially coupled in a circular circuit;	The '742 Accused Instrumentalities include at least two soft decision decoders including first and second soft decision decoders serially coupled in a circular circuit. For example, the representative product, ZenFone 7 Pro, includes first and second soft decision decoders. *See, e.g.*, 3GPP TS 26.267 at 24:  **Figure 18: Turbo decoder** In the example, the first soft decision decoder outputs soft decision that becomes "soft output 1" after exiting "interleaver." This "soft output 1" is fed as input into the second soft decision decoder, "2nd constituent decoder," for the second decision decoder to process. The second soft decision decoder also

	receives the "soft parity 2, ptail 2" input signal. The second soft decision decoder outputs soft decision that becomes "soft output 2" after exiting "deinterleaver." This "soft output 2" is fed as input into the first soft decision decoder for the first soft decision decoder to process. The first soft decision decoder, "1st constituent decoder," also receives the "soft data" and "soft pairty 1, ptail 1" input signals. *See also Cheng et. al.* "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y: "Figure 3 shows the system architecture. The blocks in the dashed box handle the turbo decoding operations, and those outside the dashed box belong to the DVFS scheme. Turbo decoding is an iterative process with several turbo iterations. Each turbo iteration comprises two soft-in, soft-out (SISO) decoding processes using BCJR algorithm [8]with the first one performed on the input code block in the original order and the second one in an order generated by the interleaver block." *See also* Valenti et al., "UMTS Turbo Code and an Efficient Decoder," International Journal of Wireless Information Networks, Vol. 8, No. 4 (Oct. 2001), at 206, *available at* https://community.wvu.edu/~mcvalenti/documents/valenti01.pdf Page 206, section 5. THE MAX* OPERATOR, 5.1. Log-MAP Algorithm, 5.2. Max-log-MAP Algorithm *See also* ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

	`` /*==*/ /* DECODER FUNCTION: Bcjr */ /*--*/ /* Description: BCJR algorithm */ /* */ /* In: const IntLLR* parity -> received parity soft bits */ /* InOut: IntLLR* extrinsic <-> extrinsic information */ /*--*/ void Bcjr(const IntLLR *parity, IntLLR *extrinsic) ```
wherein each decoder processes soft decision from the preceding decoder output data in an iterative mode;	Each decoder of the '742 Accused Instrumentalities processes soft decision from the preceding decoder output data in an iterative mode. For example, in the representative product ZenFone 7 Pro, each decoder processes soft decision from the preceding decoder output data in an iterative mode: "The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies…" *See* Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf. *See also Cheng et. al.* "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y: "Figure 3 shows the system architecture. The blocks in the dashed box handle the turbo decoding operations, and those outside the dashed box belong to the DVFS scheme. Turbo decoding is an iterative

	process with several turbo iterations. Each turbo iteration comprises two soft-in, soft-out (SISO) decoding processes using BCJR algorithm [8]with the first one performed on the input code block in the original order and the second one in an order generated by the interleaver block." *See also* ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf: • ecall-fec.c line 244 */* add received systematic bits to extrinsic information */* *for (j = 0; j < NRB_INFO_CRC+NRB_TAIL; j++)* • ecall-fec.c line 256 */* add received systematic bits to extrinsic information */* *for (j = 0; j < NRB_INFO_CRC; j++)* • ecall-fec.c line 240 */* iterative decoding */* *for (i = 0; i < FEC_ITERATIONS; i++)*
at least one memory module that is electrically coupled to an output of a corresponding soft decision decoder, wherein the output of the memory module associated with a last soft decision decoder is fed back as an input to the first soft decision decoder,	In the '742 Accused Instrumentalities, at least one memory module that is electrically coupled to an output of a corresponding soft decision decoder, wherein the output of the memory module associated with a last soft decision decoder is fed back as an input to the first soft decision decoder. For example, the representative product, ZenFone 7 Pro, includes at least one memory module (e.g., interleaver" to the right of the "1st constituent decoder" and "deinterleaver" in the figure, that is electrically coupled to an output of a corresponding soft decision decoder (e.g., "1st constituent decoder" and "2nd constituent decoder")), wherein the output of the memory module associated with the second soft decision decoder ("deinterleaver") is fed back as an input of the first soft decision decoder, as shown below:

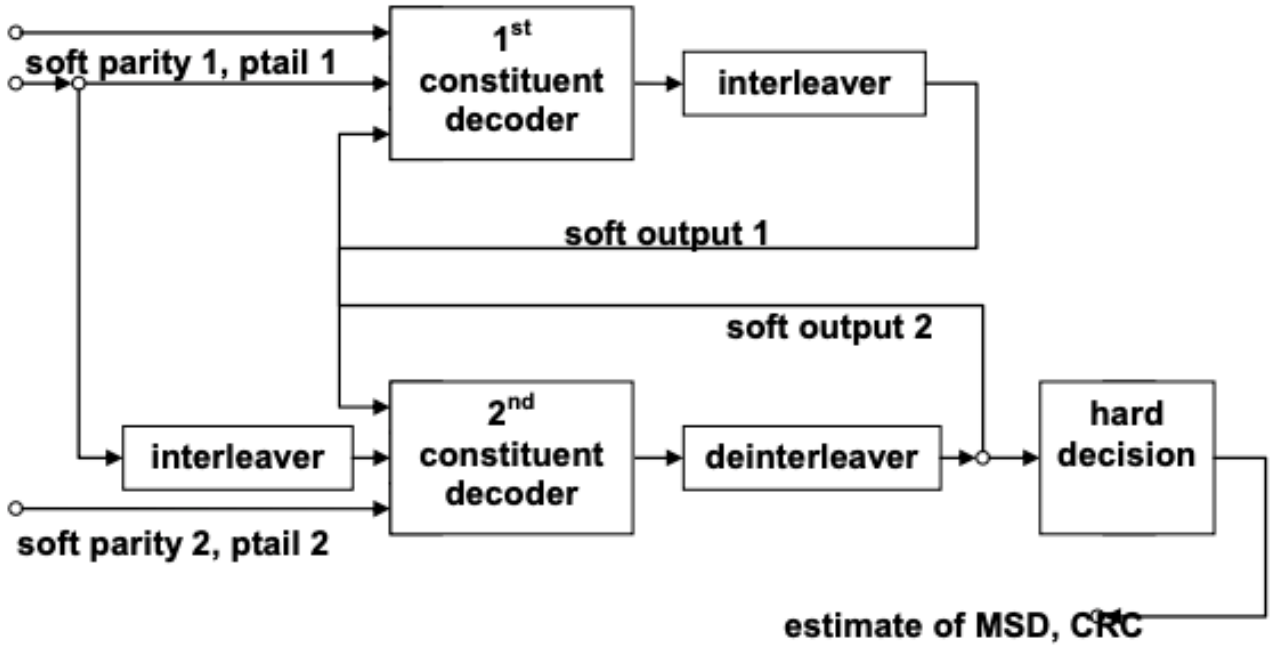


Figure 18: Turbo decoder

See 3GPP TS 26.267, at 25.

In the example, the "deinterleaver" memory module is associated with the second soft decision decoder, "2nd constituent decoder." The output of the deinterleaver, "soft output 2," is fed back as an input of the first soft decision decoder, "1st constituent decoder."

Additional evidence that the “interleaver” and “deinterleaver” comprise memory modules is shown in source code associated with the figure:

```
/* initialize memory */
  Le12 = (IntLLR*)&decBits[0];
  Le21 = (IntLLR*)&decBits[sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL)];
```

```
memset(Le12, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));
memset(Le21, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

/* iterative decoding */
for (i = 0; i < FEC_ITERATIONS; i++) {
 memcpy(Le12, Le21, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

 /* add received systematic bits to extrinsic information */
 for (j = 0; j < NRB_INFO_CRC+NRB_TAIL; j++) {
  temp = (Int32)Le12[j] + (Int32)syst1[j];
  Le12[j] = (ABS(temp) < LLR_MAX) ?
   (IntLLR)temp : (IntLLR)(SIGN(temp)*LLR_MAX);
 }
 /* decode code one (produces Le12) */
 Bcjr(parity1, Le12);

 /* interleave extrinsic information (produces interleaved Le12) */
 Interleave(Le12, Le21);

 /* add received systematic bits to extrinsic information */
 for (j = 0; j < NRB_INFO_CRC; j++) {
  temp = (Int32)Le21[j] + (Int32)syst1[interleaverSeq[j]];
  Le21[j] = (ABS(temp) < LLR_MAX) ?
   (IntLLR)temp : (IntLLR)((SIGN(temp))*LLR_MAX);
 }
 for (j = 0; j < NRB_TAIL; j++) {
  Le21[j+NRB_INFO_CRC] = syst2[j];
 }
 /* decode code two (produces interleaved Le21) */
 Bcjr(parity2, Le21);

 /* deinterleave extrinsic information (produces Le21) */
 Deinterleave(Le21);
```

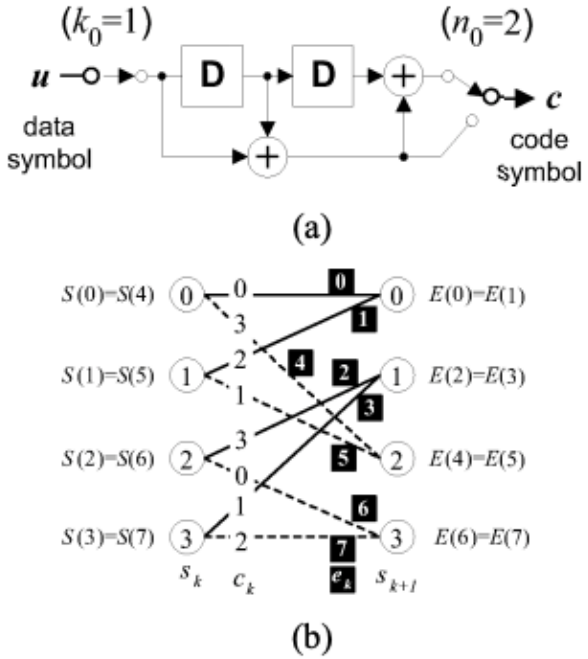
}

See, e.g., ecall-fec.c, lines 232-268, 3GPP TS 26.268, at 21.

See also Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003):

"Fig. 1. A (2, 1, 3) convolutional code. (a) An encoder with 2 memory delay elements (D) and modulo 2 adders, data symbol alphabet {0, 1}, code symbol alphabet {0, 1, 2, 3}, memory states {0, 1, 2, 3}, and code rate R = (1=2). (b) A trellis section where solid edges correspond to u = 0, and dashed edges correspond to u = 1. The output code symbols c are shown on the edges. The edges are numbered with black squares, and the edge starting and ending states are shown on the left and right, respectively."

See also id. at FIG. 1:



	See also Valenti et al., "UMTS Turbo Code and an Efficient Decoder," International Journal of Wireless Information Networks, Vol. 8, No. 4 (Oct. 2001), at 207, *available at* https://community.wvu.edu/~mcvalenti/documents/valenti01.pdf "Two key observations should be pointed out before going into the details of the algorithm: (1) It does not matter whether the forward sweep or the reverse sweep is performed first; and (2) while the partial path metrics for the entire first sweep (forward or backward) must be stored in memory, they do not need to be stored for the entire second sweep. This is because the LLR values can be computed during the second sweep, and thus partial path metrics for only two stages of the trellis (the current and previous stages) must be maintained during the second sweep." *See also Cheng et. al.* "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at Fig. 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y:

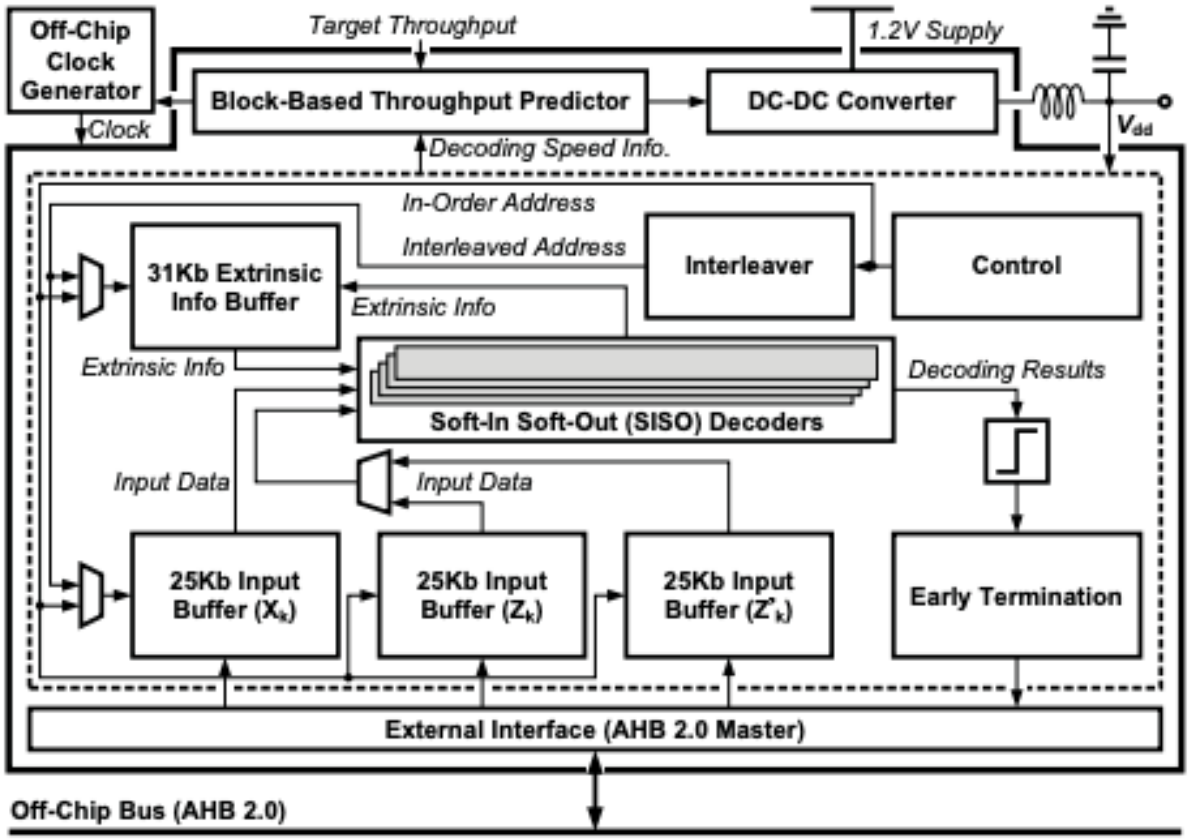
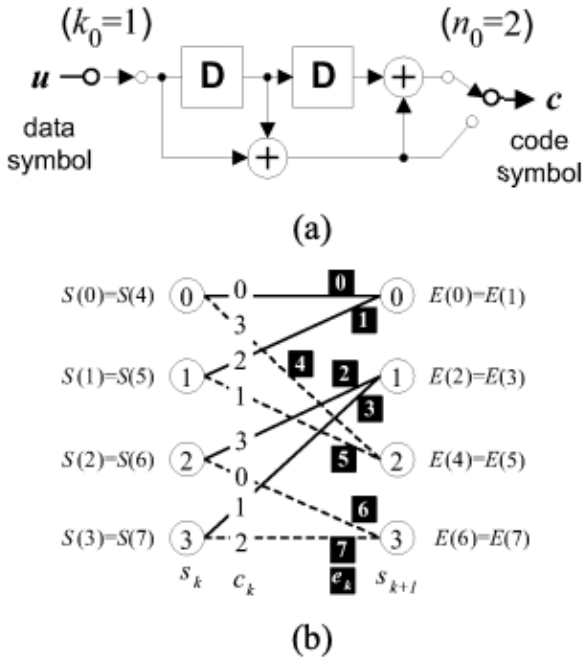


Fig. 3. The system architecture.

See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

- ecall-fec.c line 232
  / initialize memory */*

	Le12 = (IntLLR)&decBits[0];* *Le21 = (IntLLR*)&decBits[sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL)];* *memset(Le12, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));* *memset(Le21, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));* • ecall-fec.c line 250 *Bcjr(parity1, Le12); /*corresponding memory module Le12*/* • ecall-fec.c line 265 Bcjr(parity2, Le21); /*corresponding memory module Le21*/
wherein the output of the memory module associated with the first soft decision decoder is fed as an input to the second soft decision decoder, wherein the last soft decision decoder receives output of the memory module associated with the preceding soft decision decoder; and	In the '742 Accused Instrumentalities, the output of the memory module associated with the first soft decision decoder is fed as an input to the second soft decision decoder, wherein the last soft decision decoder receives output of the memory module associated with the preceding soft decision decoder. For example, in the representative product ZenFone 7 Pro, the output of the memory module associated with the first soft decision decoder is fed as an input to the second soft decision decoder, wherein the last soft decision decoder receives output of the memory module associated with the preceding soft decision decoder, as shown below: "Fig. 1. A (2, 1, 3) convolutional code. (a) An encoder with 2 memory delay elements (D) and modulo 2 adders, data symbol alphabet {0, 1}, code symbol alphabet {0, 1, 2, 3}, memory states {0, 1, 2, 3}, and code rate R = (1=2). (b) A trellis section where solid edges correspond to u = 0, and dashed edges correspond to u = 1. The output code symbols c are shown on the edges. The edges are numbered with black squares, and the edge starting and ending states are shown on the left and right, respectively." *See* Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003); *see also id.* at FIG. 1:



See also Valenti et al., "UMTS Turbo Code and an Efficient Decoder," International Journal of Wireless Information Networks, Vol. 8, No. 4 (Oct. 2001), at 207, *available at* https://community.wvu.edu/~mcvalenti/documents/valenti01.pdf

"Two key observations should be pointed out before going into the details of the algorithm: (1) It does not matter whether the forward sweep or the reverse sweep is performed first; and (2) while the partial path metrics for the entire first sweep (forward or backward) must be stored in memory, they do not need to be stored for the entire second sweep. This is because the LLR values can be computed during the second sweep, and thus partial path metrics for only two stages of the trellis (the current and previous stages) must be maintained during the second sweep."

See also Cheng et. al. "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at Fig. 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y:

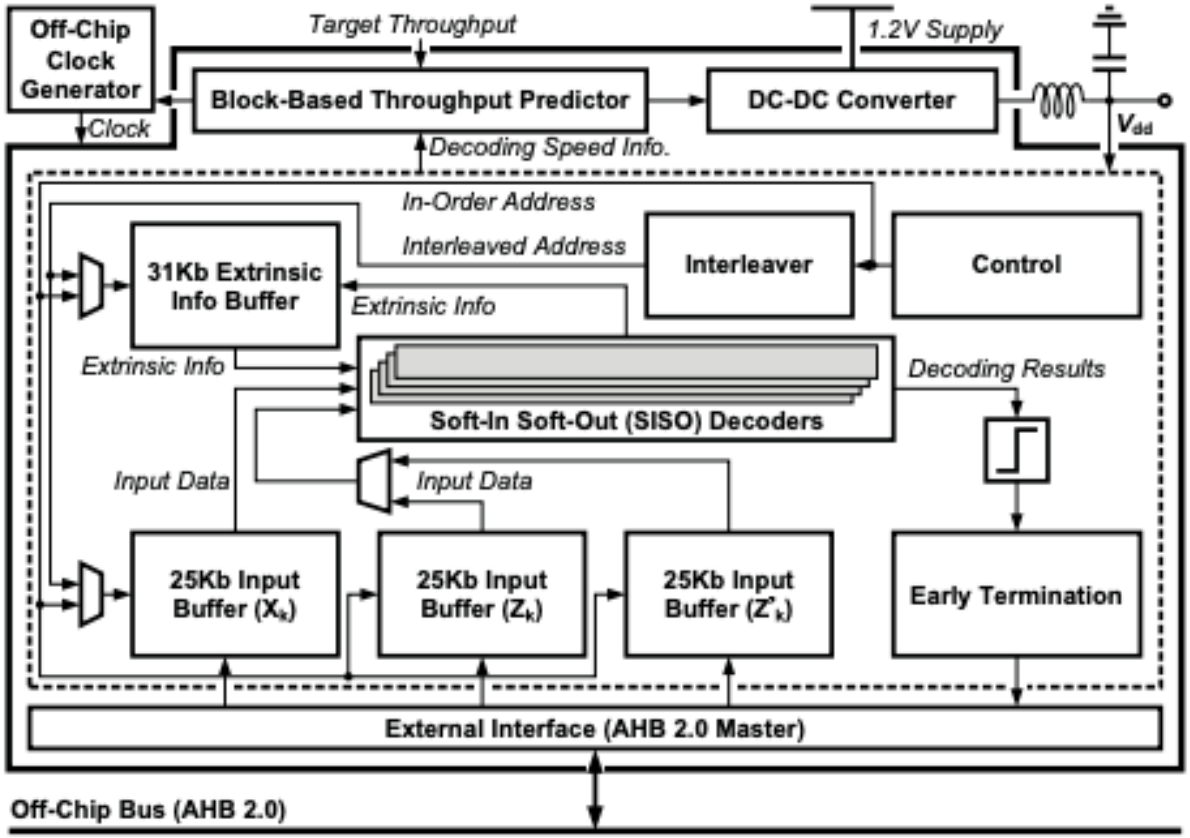


Fig. 3. The system architecture.

	See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf: • ecall-fec.c line 241 *memcpy(Le12, Le21, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));* • ecall-fec.c line 252 */* interleave extrinsic information (produces interleaved Le12) */* *Interleave(Le12, Le21);*
wherein the first soft decision decoder further receives the first and second shifted input signals from the input buffer and the second soft decision decoder further receives the third shifted input signals from the input buffer.	In the '742 Accused Instrumentalities, the first soft decision decoder further receives the first and second shifted input signals from the input buffer and the second soft decision decoder further receives the third shifted input signals from the input buffer. For example, in the representative product ZenFone 7 Pro, the first soft decision decoder further receives the first and second shifted input signals from the input buffer and the second soft decision decoder further receives the third shifted input signals from the input buffer, as shown below: "Two key observations should be pointed out before going into the details of the algorithm: (1) It does not matter whether the forward sweep or the reverse sweep is performed first; and (2) while the partial path metrics for the entire first sweep (forward or backward) must be stored in memory, they do not need to be stored for the entire second sweep. This is because the LLR values can be computed during the second sweep, and thus partial path metrics for only two stages of the trellis (the current and previous stages) must be maintained during the second sweep." *See* Valenti et al., "UMTS Turbo Code and an Efficient Decoder," International Journal of Wireless Information Networks, Vol. 8, No. 4 (Oct. 2001), at 207, *available at* https://community.wvu.edu/~mcvalenti/documents/valenti01.pdf *See also Cheng et. al.* "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at Fig. 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-

a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y:

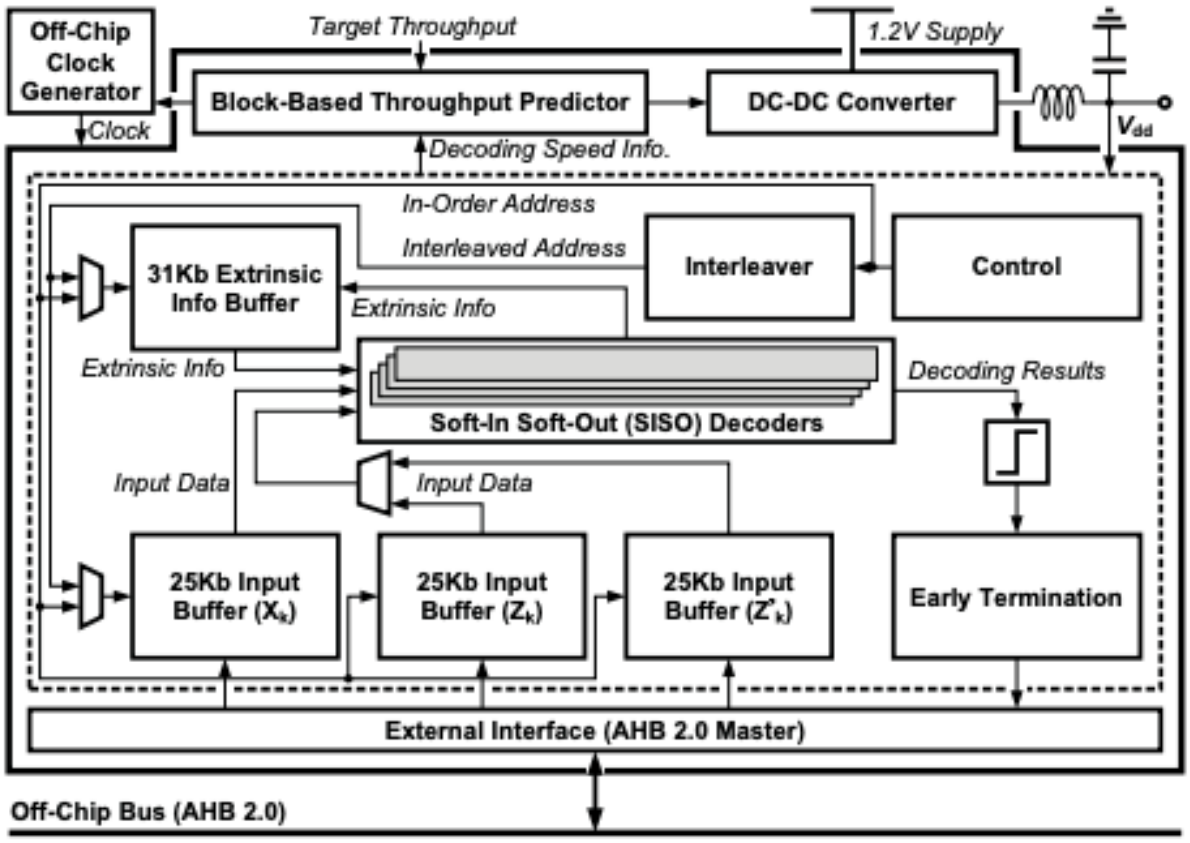


Fig. 3. The system architecture.

See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

- ecall-fec.c line 200

	*void UpdateBuffer(IntLLR *chLLRbuffer, const IntLLR *softInBits, Int16 rv)* • ecall-fec.c line 225 *void DecodeBuffer(const IntLLR *syst1, const IntLLR *syst2,* *const IntLLR *parity1, const IntLLR *parity2, Ord1 *decBits)*
Claim 6	
6. A method of iteratively decoding a plurality of sequences of received baseband signals, the method comprising:	The '742 Accused Instrumentalities provide a method of iteratively decoding a plurality of sequences of received baseband signals. For example, the representative product ZenFone 7 Pro performs a method of iteratively decoding a plurality of sequences of received baseband signals, as shown below:

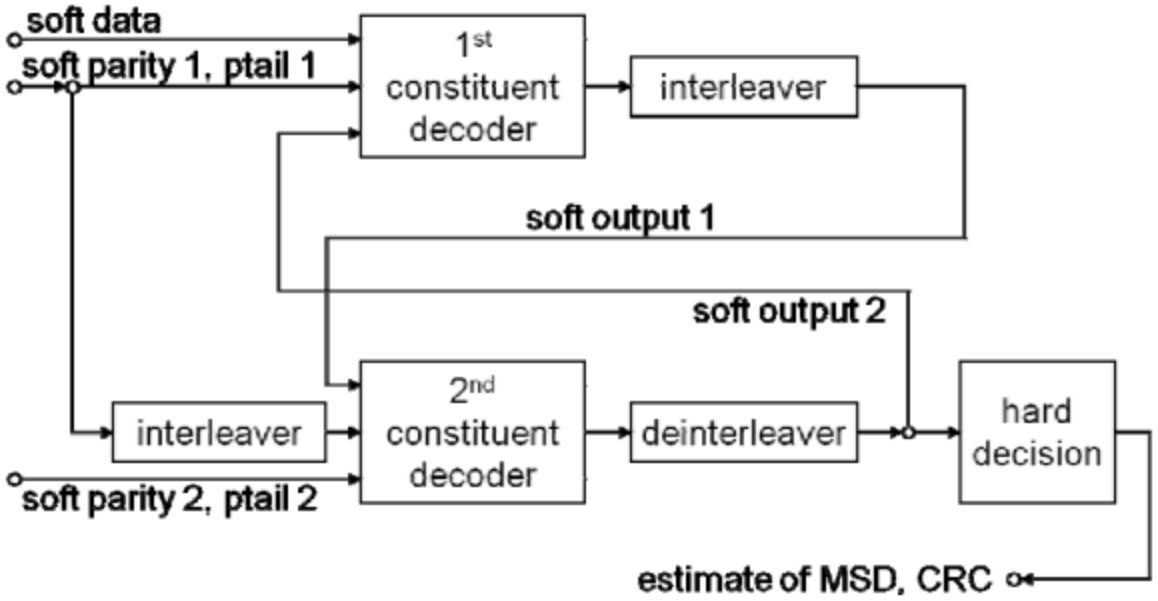


Figure 18: Turbo decoder

See 3GPP TS 26.267 at 25.

As another example, each accused instrumentality, including the ZenFone 7 Pro, uses the BCJR algorithm, which performs iterative decoding:

```
/*============================================================================*/
/* DECODER FUNCTION: Bcjr                                                     */
/*----------------------------------------------------------------------------*/
/* Description: BCJR algorithm                                                */
/*                                                                            */
/* In:     const IntLLR* parity      -> received parity soft bits             */
/* InOut:  IntLLR*       extrinsic  <-> extrinsic information                 */
/*----------------------------------------------------------------------------*/
void Bcjr(const IntLLR *parity, IntLLR *extrinsic)
```

See "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); eCall data transfer; In-band modem solution; ANSI-C reference code (3GPP TS 26.268 version 11.0.0 Release 11)," at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf [hereinafter 3GPP TS 26.268].

See Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf:

"The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies…"

See also Cheng et. al. "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y:

"Figure 3 shows the system architecture. The blocks in the dashed box handle the turbo decoding

operations, and those outside the dashed box belong to the DVFS scheme. Turbo decoding is an iterative process with several turbo iterations. Each turbo iteration comprises two soft-in, soft-out (SISO) decoding processes using BCJR algorithm [8]with the first one performed on the input code block in the original order and the second one in an order generated by the interleaver block."

See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 14, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

Type/Constant	Dimension	Description
/* Synchronization */		
const Int16 wakeupSin500	[16]	sine waveform at 500 Hz
const Int16 wakeupCos500	[16]	cosine waveform at 500 Hz
const Int16 wakeupSin800	[10]	sine waveform at 800 Hz
const Int16 wakeupCos800	[10]	cosine waveform at 800 Hz

See also id. at 20:

```
/*============================================================================*/
/* PSAP FUNCTION: PsapReceiver                                                */
/*----------------------------------------------------------------------------*/
/* Description: PSAP receiver function (decoding is done outside)             */
/*                                                                            */
/* In:     const ModState* ms        -> modulator struct                      */
/*         const Int16*    pcm       -> input data for demodulation           */
/* Out:    IntLLR*         softBits  <- demodulated soft bit sequence         */
/*----------------------------------------------------------------------------*/
void PsapReceiver(const ModState *ms, const Int16 *pcm, IntLLR *softBits)


/*============================================================================*/
/* PSAP FUNCTION: SymbolDemod                                                 */
/*----------------------------------------------------------------------------*/
/* Description: symbol demodulator                                            */
/*                                                                            */
/* In:     const ModState* ms        -> modulator struct                      */
/*         const Int16*    mPulse    -> received pulse train                  */
/* Out:    IntLLR*         softBits  <- demodulated soft bit sequence         */
/*----------------------------------------------------------------------------*/
void SymbolDemod(const ModState *ms, const Int16 *mPulse, IntLLR *softBits)
```

- ecall-fec.c line 163
  *Bool FecDecode(const IntLLR *in, Int16 rv, Ord1 *out)*

```
/*============================================================================*/
/* DECODER FUNCTION: FecDecode                                                */
/*----------------------------------------------------------------------------*/
/* Description: decoding to find the MSD                                      */
/*                                                                            */
/* In:     const IntLLR* in   -> received soft bits                           */
/*         Int16         rv   -> redundancy version                           */
/* Out:    Ord1*         out  <- decoded MSD in binary representation         */
/* Return: Bool                <- result of CRC check                         */
/*----------------------------------------------------------------------------*/
Bool FecDecode(const IntLLR *in, Int16 rv, Ord1 *out)
```

	• ecall-fec.c line 240 */*iterative decoding*/* *for (i = 0; i < FEC_ITERATIONS; i++)* *See also* May et al., "A 150Mbit/s 3GPP LTE Turbo code decoder," 2010 Design, Automation & Test in Europe Conference & Exhibition (March 8-12, 2010), *available at* https://ieeexplore.ieee.org/document/5457035/authors#authors: "3GPP long term evolution (LTE) enhances the wireless communication standards UMTS and HSDPA towards higher throughput. A throughput of 150 Mbit/s is specified for LTE using 2×2 MIMO. For this, highly punctured Turbo codes with rates up to 0.95 are used for channel coding, which is a big challenge for decoder design. This paper investigates efficient decoder architectures for highly punctured LTE Turbo codes. We present a 150 Mbit/s 3GPP LTE Turbo code decoder, which is part of an industrial SDR multi-standard baseband processor chip."
providing an input buffer comprising at least three shift registers, for receiving an input signal and generating first, second, and third shifted input signals;	The '742 Accused Instrumentalities provide an input buffer comprising at least three shift registers, for receiving an input signal and generating first, second, and third shifted input signals. For example, the representative product ZenFone 7 Pro provides (e.g., via an input buffer) input to the constituent decoders of the turbo decoder. The input buffer comprises at least three shift registers. The input buffer receives an input signal, and first, second, and third shifted input signals are generated for input to a turbo decoder. The generated first, second, and third shifted input signals, shown as "soft data," "soft parity 1, ptail 1," and "soft parity 2, ptail 2," are input into the 1st constituent decoder and 2nd constituent decoder as shown below:

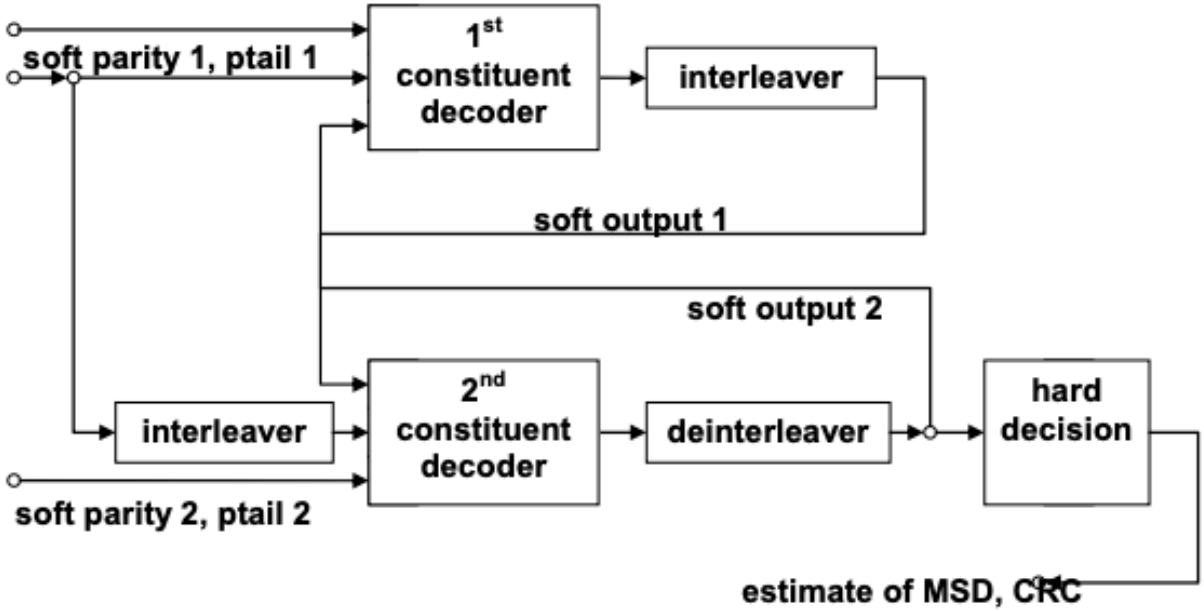


Figure 18: Turbo decoder

See 3GPP TS 26.267, at 25.

Each such buffer provides three sections based on operations of a turbo encoder. For example, an input buffer is denoted as Figure 7, labeled as a "channel coded bit buffer":

Figure 7: Channel coded bit buffer

	Note that the "soft data" of Figure 18 corresponds to the "MSD+CRC," "tail 1," and "tail 2" fields of Figure 7. *See* 3GPP TS 26.267, at 14. Such a channel coded bit buffer can then be decoded using shifting to generate the first, second, and third shifted input signals, which are stored in registers for input into the turbo decoder: "6.1.3 Modulation The encoded binary data stream bits b_i are grouped into symbols. Each symbol d_j carries 4 bits of information and modulates one basic downlink waveform. … Table 4 describes the symbol modulation mapping between symbol and the downlink waveform. The downlink waveform is derived from the basic downlink waveform $p_{DL}(n)$ by a cyclic right-shift by k samples, denoted by (p → k) , and multiplication with a sign q." *See* 3GPP TS 26.267, at 22. *See also Cheng et. al.* "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at Fig. 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y :

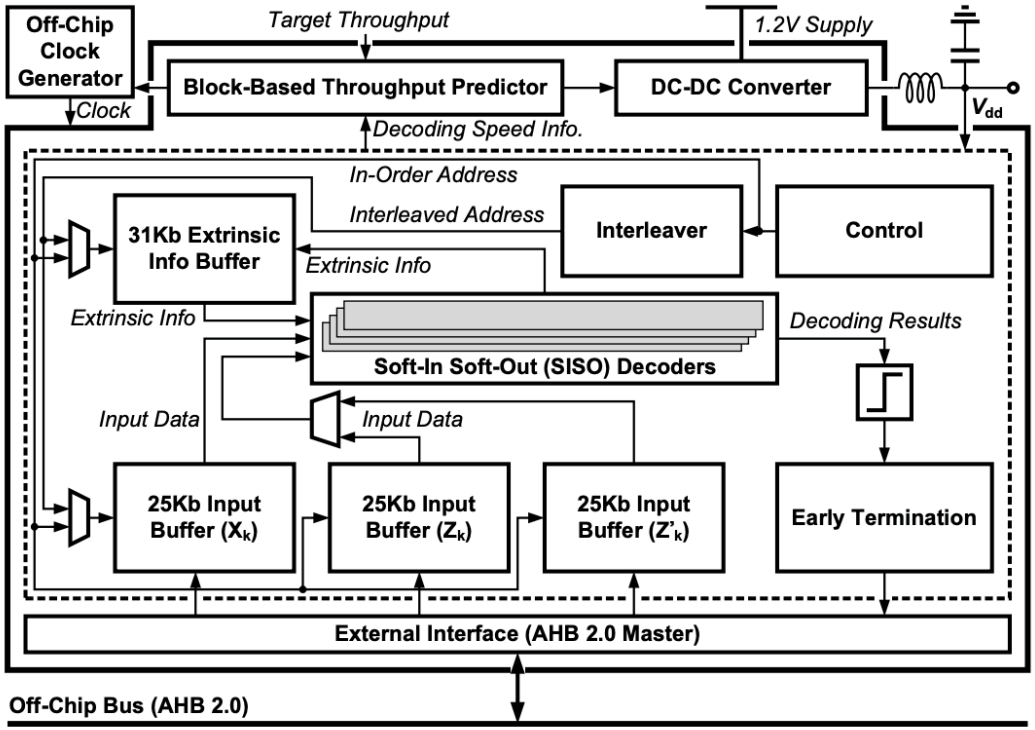


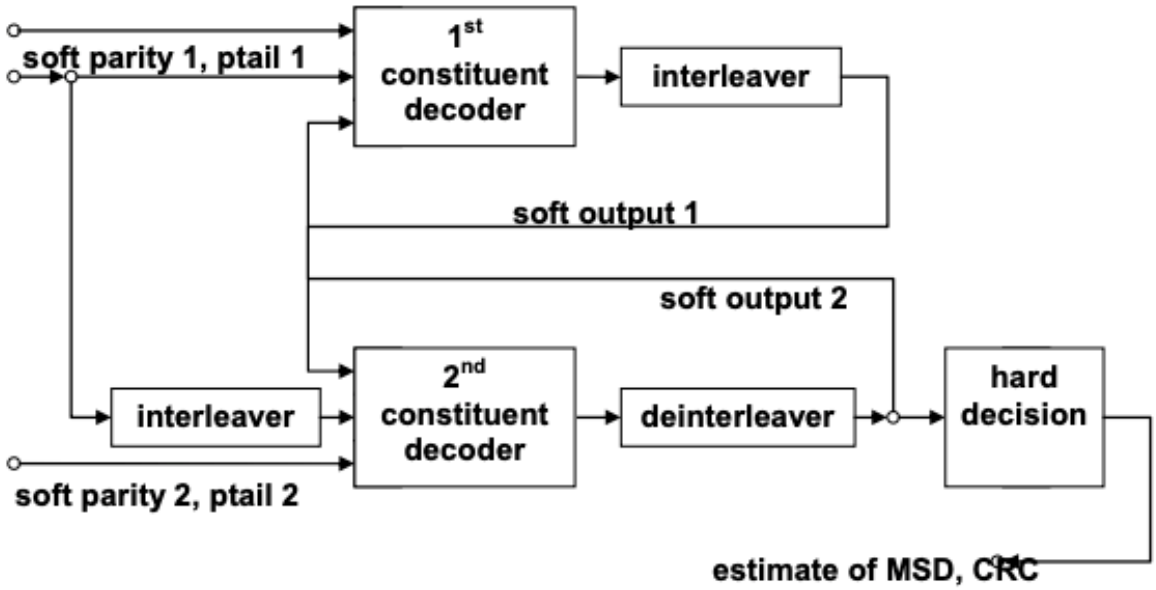
Fig. 3. The system architecture.

See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

- ecall-fec.c line 200
  *void UpdateBuffer(IntLLR *chLLRbuffer, const IntLLR *softInBits, Int16 rv)*

- ecall-fec.c line 225
  *void DecodeBuffer(const IntLLR *syst1, const IntLLR *syst2,*

CLAIM CHART
U.S. Patent No. 6,813,742

	*const IntLLR *parity1, const IntLLR *parity2, Ord1 *decBits)*
providing first and second soft decision decoders serially coupled in a circular circuit, wherein each decoder processes soft decision from the preceding decoder output data, and wherein the first decoder further receives the first and second shifted input signals from the input buffer and the second decoder further receives the third shifted input signal from the input buffer;	The '742 Accused Instrumentalities provide first and second soft decision decoders serially coupled in a circular circuit, wherein each decoder processes soft decision from the preceding decoder output data, and wherein the first decoder further receives the first and second shifted input signals from the input buffer and the second decoder further receives the third shifted input signal from the input buffer. For example, the representative product, ZenFone 7 Pro, includes first and second soft decision decoders. *See, e.g.,* 3GPP TS 26.267 at 25:  **Figure 18: Turbo decoder** In the example, the first soft decision decoder outputs soft decision that becomes "soft output 1" after exiting "interleaver." This "soft output 1" is fed as input into the second soft decision decoder, "2nd

	constituent decoder," for the second decision decoder to process. The second soft decision decoder also receives the "soft parity 2, ptail 2" input signal. The second soft decision decoder outputs soft decision that becomes "soft output 2" after exiting "deinterleaver." This "soft output 2" is fed as input into the first soft decision decoder for the first soft decision decoder to process. The first soft decision decoder, "1st constituent decoder," also receives the "soft data" and "soft pairty 1, ptail 1" input signals. *See also Cheng et. al.* "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y: "Figure 3 shows the system architecture. The blocks in the dashed box handle the turbo decoding operations, and those outside the dashed box belong to the DVFS scheme. Turbo decoding is an iterative process with several turbo iterations. Each turbo iteration comprises two soft-in, soft-out (SISO) decoding processes using BCJR algorithm [8]with the first one performed on the input code block in the original order and the second one in an order generated by the interleaver block." *See also* Valenti et al., "UMTS Turbo Code and an Efficient Decoder," International Journal of Wireless Information Networks, Vol. 8, No. 4 (Oct. 2001), at 206, *available at* https://community.wvu.edu/~mcvalenti/documents/valenti01.pdf Page 206, section 5. THE MAX* OPERATOR, 5.1. Log-MAP Algorithm, 5.2. Max-log-MAP Algorithm *See also* ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

	`` /*==*/ /* DECODER FUNCTION: Bcjr */ /*--*/ /* Description: BCJR algorithm */ /* */ /* In: const IntLLR* parity -> received parity soft bits */ /* InOut: IntLLR* extrinsic <-> extrinsic information */ /*--*/ void Bcjr(const IntLLR *parity, IntLLR *extrinsic) ```
providing at least one memory module coupled to an output of each of the first and second soft decision decoders, wherein the output of the memory module associated with the second soft decision decoder is fed back as an input of the first soft decision decoder;	The '742 Accused Instrumentalities provide at least one memory module coupled to an output of each of the first and second soft decision decoders, wherein the output of the memory module associated with the second soft decision decoder is fed back as an input of the first soft decision decoder. For example, the representative product, ZenFone 7 Pro, includes at least one memory module (e.g., interleaver" to the right of the "1st constituent decoder" and "deinterleaver" in the figure, that is electrically coupled to an output of a corresponding soft decision decoder (e.g., "1st constituent decoder" and "2nd constituent decoder")), wherein the output of the memory module associated with the second soft decision decoder ("deinterleaver") is fed back as an input of the first soft decision decoder, as shown below:

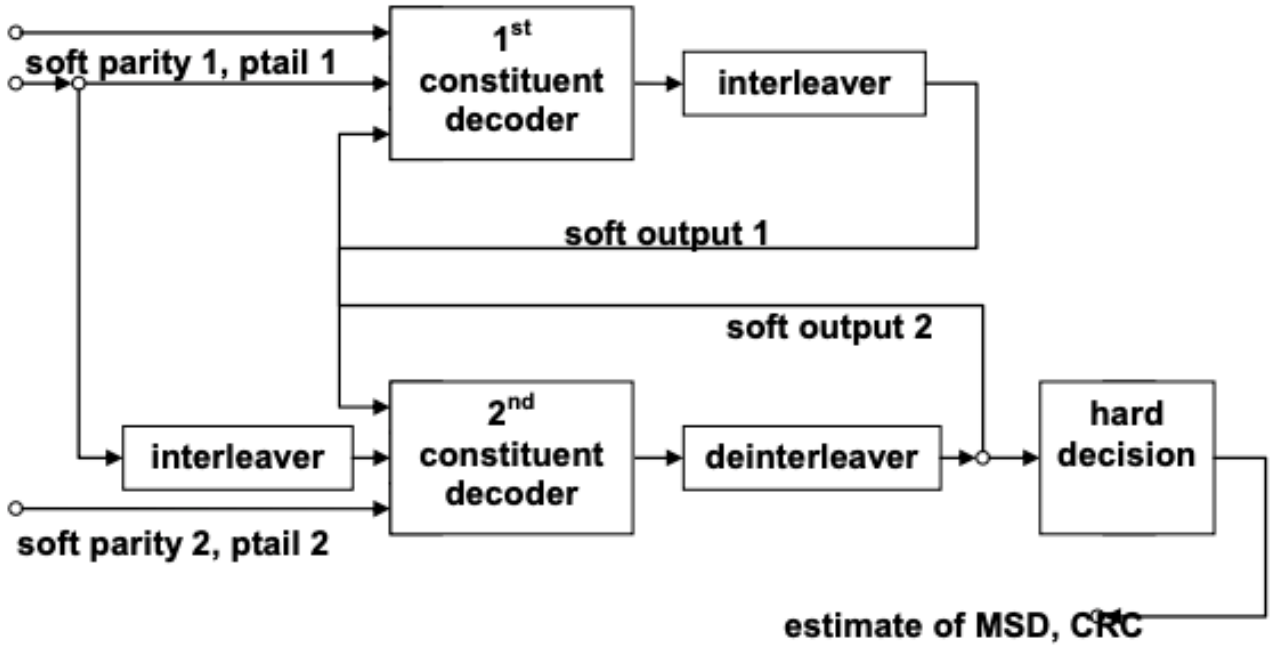


Figure 18: Turbo decoder

See 3GPP TS 26.267, at 25.

In the example, the "deinterleaver" memory module is associated with the second soft decision decoder, "2nd constituent decoder." The output of the deinterleaver, "soft output 2," is fed back as an input of the first soft decision decoder, "1st constituent decoder."

Additional evidence that the “interleaver” and “deinterleaver” comprise memory modules is shown in source code associated with the figure:

```
/* initialize memory */
  Le12 = (IntLLR*)&decBits[0];
  Le21 = (IntLLR*)&decBits[sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL)];
```

```
memset(Le12, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));
memset(Le21, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

/* iterative decoding */
for (i = 0; i < FEC_ITERATIONS; i++) {
 memcpy(Le12, Le21, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

 /* add received systematic bits to extrinsic information */
 for (j = 0; j < NRB_INFO_CRC+NRB_TAIL; j++) {
  temp = (Int32)Le12[j] + (Int32)syst1[j];
  Le12[j] = (ABS(temp) < LLR_MAX) ?
   (IntLLR)temp : (IntLLR)(SIGN(temp)*LLR_MAX);
 }
 /* decode code one (produces Le12) */
 Bcjr(parity1, Le12);

 /* interleave extrinsic information (produces interleaved Le12) */
 Interleave(Le12, Le21);

 /* add received systematic bits to extrinsic information */
 for (j = 0; j < NRB_INFO_CRC; j++) {
  temp = (Int32)Le21[j] + (Int32)syst1[interleaverSeq[j]];
  Le21[j] = (ABS(temp) < LLR_MAX) ?
   (IntLLR)temp : (IntLLR)((SIGN(temp))*LLR_MAX);
 }
 for (j = 0; j < NRB_TAIL; j++) {
  Le21[j+NRB_INFO_CRC] = syst2[j];
 }
 /* decode code two (produces interleaved Le21) */
 Bcjr(parity2, Le21);

 /* deinterleave extrinsic information (produces Le21) */
 Deinterleave(Le21);
```

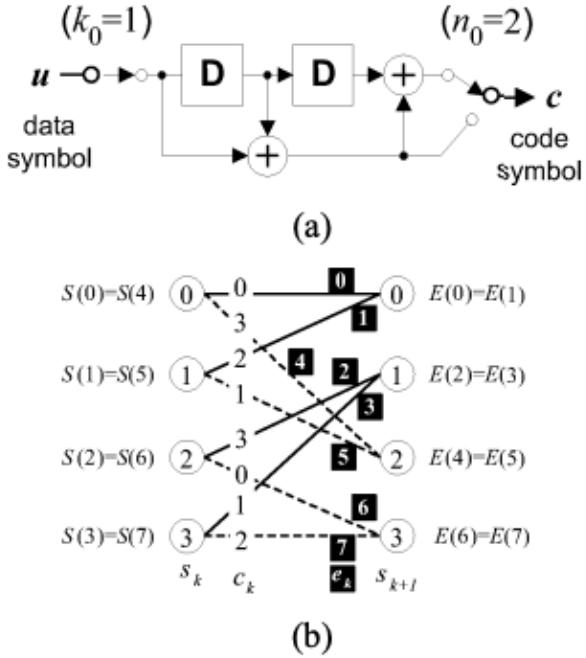
}

See, e.g., ecall-fec.c, lines 232-268, 3GPP TS 26.268, at 21.

See also Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003):

"Fig. 1. A (2, 1, 3) convolutional code. (a) An encoder with 2 memory delay elements (D) and modulo 2 adders, data symbol alphabet {0, 1}, code symbol alphabet {0, 1, 2, 3}, memory states {0, 1, 2, 3}, and code rate R = (1=2). (b) A trellis section where solid edges correspond to u = 0, and dashed edges correspond to u = 1. The output code symbols c are shown on the edges. The edges are numbered with black squares, and the edge starting and ending states are shown on the left and right, respectively."

See also id. at FIG. 1:



	See also Valenti et al., "UMTS Turbo Code and an Efficient Decoder," International Journal of Wireless Information Networks, Vol. 8, No. 4 (Oct. 2001), at 207, *available at* https://community.wvu.edu/~mcvalenti/documents/valenti01.pdf "Two key observations should be pointed out before going into the details of the algorithm: (1) It does not matter whether the forward sweep or the reverse sweep is performed first; and (2) while the partial path metrics for the entire first sweep (forward or backward) must be stored in memory, they do not need to be stored for the entire second sweep. This is because the LLR values can be computed during the second sweep, and thus partial path metrics for only two stages of the trellis (the current and previous stages) must be maintained during the second sweep." *See also Cheng et. al.* "A 0.077 to 0.168 nj/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," 2010 IEEE Custom Integrated Circuits Conference (CICC) (19-22 Sept. 2010), at Fig. 3, *available at* https://dspace.mit.edu/bitstream/handle/1721.1/72198/Chandrakasan-a%200.077%20to%200.168.pdf?sequence=1&isAllowed=y:

CLAIM CHART
U.S. Patent No. 6,813,742

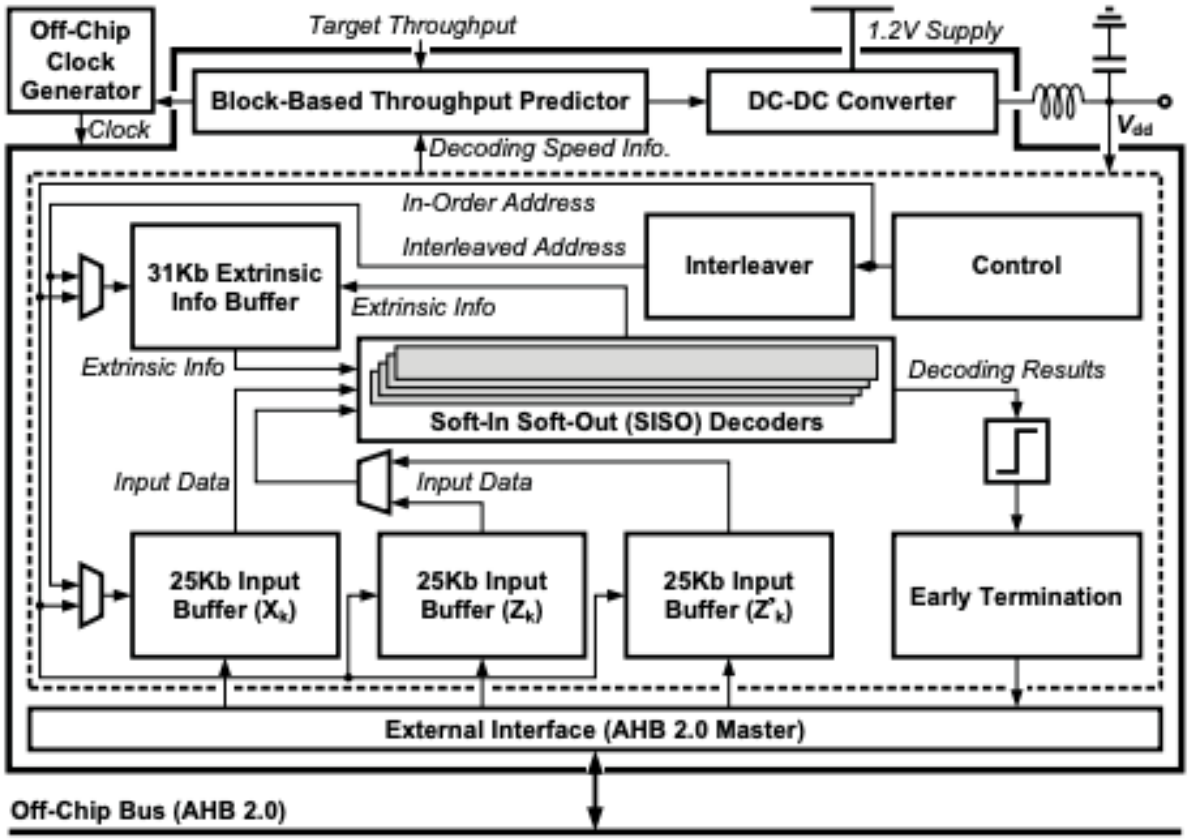


Fig. 3. The system architecture.

See also ETSI 3GPP TS 126 268 V11.0.0 (2012-10), at 21, *available at* https://www.etsi.org/deliver/etsi_ts/126200_126299/126268/11.00.00_60/ts_126268v110000p.pdf:

- ecall-fec.c line 232
  / initialize memory */*

	Le12 = (IntLLR)&decBits[0];* *Le21 = (IntLLR*)&decBits[sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL)];* *memset(Le12, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));* *memset(Le21, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));* • ecall-fec.c line 250 *Bcjr(parity1, Le12); /*corresponding memory module Le12*/* • ecall-fec.c line 265 Bcjr(parity2, Le21); /*corresponding memory module Le21*/
processing systematic information data and extrinsic information data using the maximum a posteriori (MAP) probability algorithm, and/or logarithm approximation algorithm;	The '742 Accused Instrumentalities process systematic information data and extrinsic information data using the maximum a posteriori (MAP) probability algorithm, and/or logarithm approximation algorithm. For example, the representative product, the ZenFone 7 Pro processes systematic information data and extrinsic information data using the maximum a posteriori (MAP) probability algorithm, and/or logarithm approximation algorithm. The ZenFone 7 Pro, for example, uses at least the BCJR algorithm for turbo decoding in accordance with the figure below:

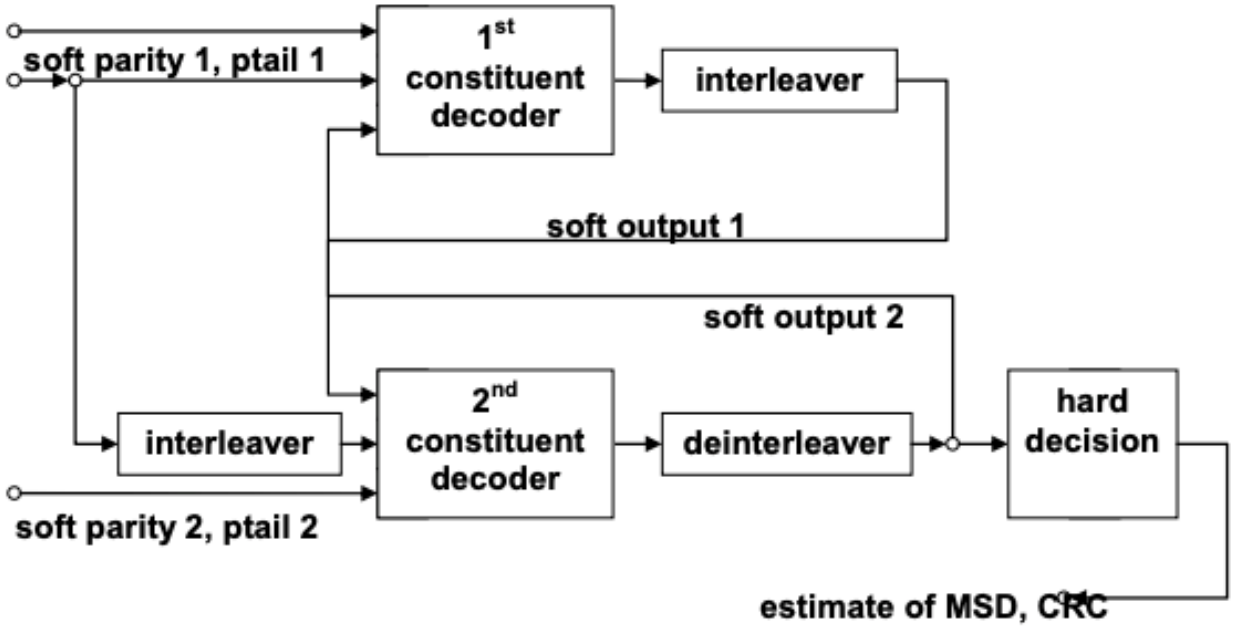


Figure 18: Turbo decoder

See 3GPP TS 26.267, at 25.

See also Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf:

"Turbo codes are composed of an interconnection of component codes through interleavers, typically convolutional codes, and their decoders consist of an equal number of component decoders each of which operates on its corresponding codeword and shares information with other component decoders iteratively according to the topology of the encoder. The decoding algorithm in the component decoders is the maximum a-posteriori probability (MAP) algorithm typically implemented in the form known as the Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [6]. The main advantage of a MAP decoding algorithm over a maximum likelihood decoding algorithm such as the Viterbi algorithm [7] is that it

produces optimum soft information which is crucial to the operation of these decoders. The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies. The advantages of the SISO-APP algorithm over other forms of the MAP algorithm is that it is independent of the code type (systematic/nonsystematic, recursive/nonrecursive, trellis with multiple edges), and it generates reliability information for code symbols as well as message symbols which makes it applicable irrespective of the concatenation scheme (parallel/serial/hybrid), and hence will be considered in this paper."

See also id. at 629 ("The decoding problem can now be defined as follows: given a noisy version of $\underline{c}$ denoted by $\underline{y}=\Delta(y_1,\cdots,y_k,\cdots,y_L)$, find the data sequence $\underline{u}$. There are two probabilistic solutions to this decoding problem. Maximum likelihood (ML) decoding determines the most likely connected path $\underline{s}$ through the trellis that maximizes the probability $P(\underline{y}|\underline{s})$. From $\underline{s}$, the most likely data sequence $\underline{u}$ is easily determined using (1). On the other hand, MAP decoding, which we consider here, determines $\underline{u}$ by estimating each of the symbols u_k independently using the observations $\underline{y}$. The kth estimated symbol u_k is the one that maximizes the posterior probability $P(u_k|\underline{y})$, and hence the name symbol-by-symbol MAP. The SISO-APP algorithm, a generalized version of the BCJR-APP algorithm [6], is a probabilistic algorithm that solves the MAP decoding problem.").

The source code associated with the figure indicates that it processes systematic information data and extrinsic information data using the BCJR algorithm:

```
/* initialize memory */
  Le12 = (IntLLR*)&decBits[0];
  Le21 = (IntLLR*)&decBits[sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL)];

  memset(Le12, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));
  memset(Le21, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

  /* iterative decoding */
  for (i = 0; i < FEC_ITERATIONS; i++) {
    memcpy(Le12, Le21, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));
```

```
/* add received systematic bits to extrinsic information */
for (j = 0; j < NRB_INFO_CRC+NRB_TAIL; j++) {
  temp = (Int32)Le12[j] + (Int32)syst1[j];
  Le12[j] = (ABS(temp) < LLR_MAX) ?
    (IntLLR)temp : (IntLLR)(SIGN(temp)*LLR_MAX);
}
/* decode code one (produces Le12) */
Bcjr(parity1, Le12);

/* interleave extrinsic information (produces interleaved Le12) */
Interleave(Le12, Le21);

/* add received systematic bits to extrinsic information */
for (j = 0; j < NRB_INFO_CRC; j++) {
  temp = (Int32)Le21[j] + (Int32)syst1[interleaverSeq[j]];
  Le21[j] = (ABS(temp) < LLR_MAX) ?
    (IntLLR)temp : (IntLLR)((SIGN(temp))*LLR_MAX);
}
for (j = 0; j < NRB_TAIL; j++) {
  Le21[j+NRB_INFO_CRC] = syst2[j];
}
/* decode code two (produces interleaved Le21) */
Bcjr(parity2, Le21);

/* deinterleave extrinsic information (produces Le21) */
Deinterleave(Le21);
}
```

See, e.g., ecall-fec.c, lines 232-268, 3GPP TS 26.268, at 21.

The BCJR algorithm is a MAP probability algorithm that processes systematic information data and extrinsic information data:

	"Turbo codes are composed of an interconnection of component codes through interleavers, typically convolutional codes, and their decoders consist of an equal number of component decoders each of which operates on its corresponding codeword and shares information with other component decoders iteratively according to the topology of the encoder. The decoding algorithm in the component decoders is the maximum a-posteriori probability (MAP) algorithm typically implemented in the form known as the Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [6]. The main advantage of a MAP decoding algorithm over a maximum likelihood decoding algorithm such as the Viterbi algorithm [7] is that it produces optimum soft information which is crucial to the operation of these decoders. The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies. The advantages of the SISO-APP algorithm over other forms of the MAP algorithm is that it is independent of the code type (systematic/nonsystematic, recursive/nonrecursive, trellis with multiple edges), and it generates reliability information for code symbols as well as message symbols which makes it applicable irrespective of the concatenation scheme (parallel/serial/hybrid), and hence will be considered in this paper." *See* Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf.
generating soft decision based on the maximum a posteriori (MAP) probability algorithm and/or logarithm approximation algorithm;	The '742 Accused Instrumentalities generate soft decision based on the maximum a posteriori (MAP) probability algorithm and/or logarithm approximation algorithm. For example, the representative product, the ZenFone 7 Pro, generates soft decision based on the maximum a posteriori (MAP) probability algorithm and/or logarithm approximation algorithm. The ZenFone 7 Pro uses at least the BCJR algorithm for decoding in accordance with the figure below:

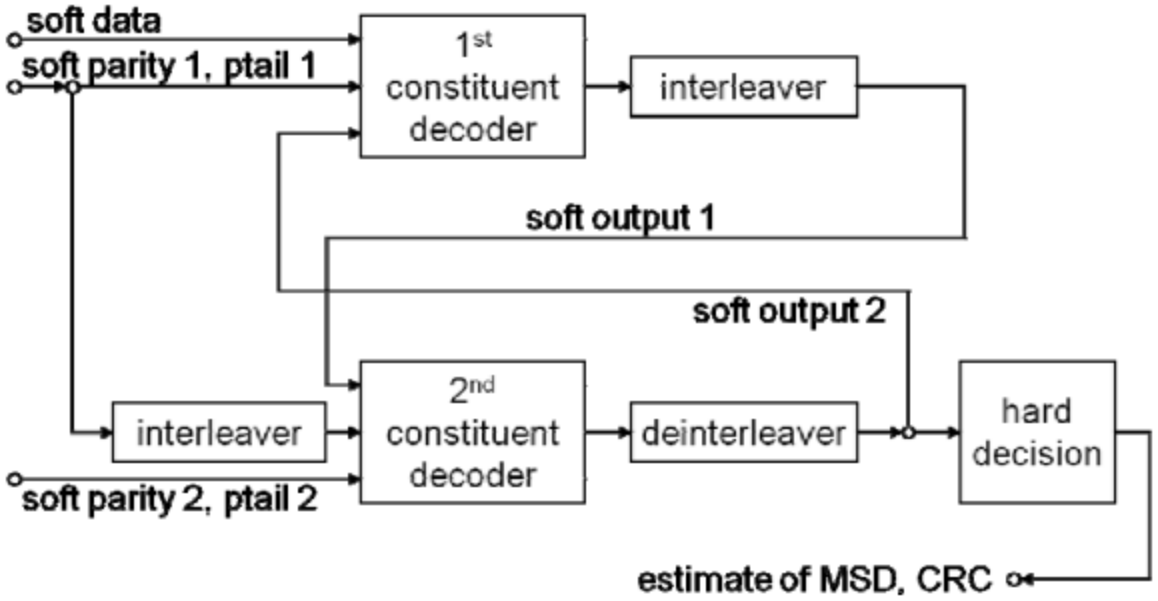


Figure 18: Turbo decoder

See 3GPP TS 26.267, at 25.

See also Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf:

"Turbo codes are composed of an interconnection of component codes through interleavers, typically convolutional codes, and their decoders consist of an equal number of component decoders each of which operates on its corresponding codeword and shares information with other component decoders iteratively according to the topology of the encoder. The decoding algorithm in the component decoders is the maximum a-posteriori probability (MAP) algorithm typically implemented in the form known as the Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [6]. The main advantage of a MAP decoding

algorithm over a maximum likelihood decoding algorithm such as the Viterbi algorithm [7] is that it produces optimum soft information which is crucial to the operation of these decoders. The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies. The advantages of the SISO-APP algorithm over other forms of the MAP algorithm is that it is independent of the code type (systematic/nonsystematic, recursive/nonrecursive, trellis with multiple edges), and it generates reliability information for code symbols as well as message symbols which makes it applicable irrespective of the concatenation scheme (parallel/serial/hybrid), and hence will be considered in this paper."

See also id. at 629 ("The decoding problem can now be defined as follows: given a noisy version of $\underline{c}$ denoted by $\underline{y}=\Delta(y_1,\cdots,y_k,\cdots,y_L)$, find the data sequence $\underline{u}$. There are two probabilistic solutions to this decoding problem. Maximum likelihood (ML) decoding determines the most likely connected path $\underline{s}$ through the trellis that maximizes the probability $P(\underline{y}|\underline{s})$. From $\underline{s}$, the most likely data sequence $\underline{u}$ is easily determined using (1). On the other hand, MAP decoding, which we consider here, determines $\underline{u}$ by estimating each of the symbols u_k independently using the observations $\underline{y}$. The kth estimated symbol u_k is the one that maximizes the posterior probability $P(u_k|\underline{y})$, and hence the name symbol-by-symbol MAP. The SISO-APP algorithm, a generalized version of the BCJR-APP algorithm [6], is a probabilistic algorithm that solves the MAP decoding problem.").

The source code associated with the figure, for example, indicates that it generates soft decision based on the BCJR algorithm. As emphasized below, the ZenFone 7 Pro uses the BCJR algorithm. Data processed by the BCJR algorithm is interleaved or deinterleaved, resulting in "soft output 1" or "soft output 2," respectively, as shown in the figure:

```
/* initialize memory */
  Le12 = (IntLLR*)&decBits[0];
  Le21 = (IntLLR*)&decBits[sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL)];

  memset(Le12, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));
  memset(Le21, 0, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

  /* iterative decoding */
```

```
for (i = 0; i < FEC_ITERATIONS; i++) {
  memcpy(Le12, Le21, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

  /* add received systematic bits to extrinsic information */
  for (j = 0; j < NRB_INFO_CRC+NRB_TAIL; j++) {
    temp = (Int32)Le12[j] + (Int32)syst1[j];
    Le12[j] = (ABS(temp) < LLR_MAX) ?
      (IntLLR)temp : (IntLLR)(SIGN(temp)*LLR_MAX);
  }
  /* decode code one (produces Le12) */
  Bcjr(parity1, Le12);

  /* interleave extrinsic information (produces interleaved Le12) */
  Interleave(Le12, Le21);

  /* add received systematic bits to extrinsic information */
  for (j = 0; j < NRB_INFO_CRC; j++) {
    temp = (Int32)Le21[j] + (Int32)syst1[interleaverSeq[j]];
    Le21[j] = (ABS(temp) < LLR_MAX) ?
      (IntLLR)temp : (IntLLR)((SIGN(temp))*LLR_MAX);
  }
  for (j = 0; j < NRB_TAIL; j++) {
    Le21[j+NRB_INFO_CRC] = syst2[j];
  }
  /* decode code two (produces interleaved Le21) */
  Bcjr(parity2, Le21);

  /* deinterleave extrinsic information (produces Le21) */
  Deinterleave(Le21);
}
```

See ecall-fec.c, lines 232-268, 3GPP TS 26.268, at 21; *see also id.*:

	• ecall-fec.c line 243 */*MAP soft decision decoder */* *void Bcjr(const IntLLR *parity, IntLLR *extrinsic)* The BCJR algorithm includes a MAP probability algorithm which processes soft input to generate soft decision: "Turbo codes are composed of an interconnection of component codes through interleavers, typically convolutional codes, and their decoders consist of an equal number of component decoders each of which operates on its corresponding codeword and shares information with other component decoders iteratively according to the topology of the encoder. The decoding algorithm in the component decoders is the maximum a-posteriori probability (MAP) algorithm typically implemented in the form known as the Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [6]. The main advantage of a MAP decoding algorithm over a maximum likelihood decoding algorithm such as the Viterbi algorithm [7] is that it produces optimum soft information which is crucial to the operation of these decoders. The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies. The advantages of the SISO-APP algorithm over other forms of the MAP algorithm is that it is independent of the code type (systematic/nonsystematic, recursive/nonrecursive, trellis with multiple edges), and it generates reliability information for code symbols as well as message symbols which makes it applicable irrespective of the concatenation scheme (parallel/serial/hybrid), and hence will be considered in this paper." *See* Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf.
weighing and storing soft decision information into the corresponding memory module;	The '742 Accused Instrumentalities weigh and store soft decision information into the corresponding memory module. For example, the representative product, the ZenFone 7 Pro, weights and stores soft decision information into the corresponding memory module, "interleaver" or "deinterleaver," as shown in the

figure:

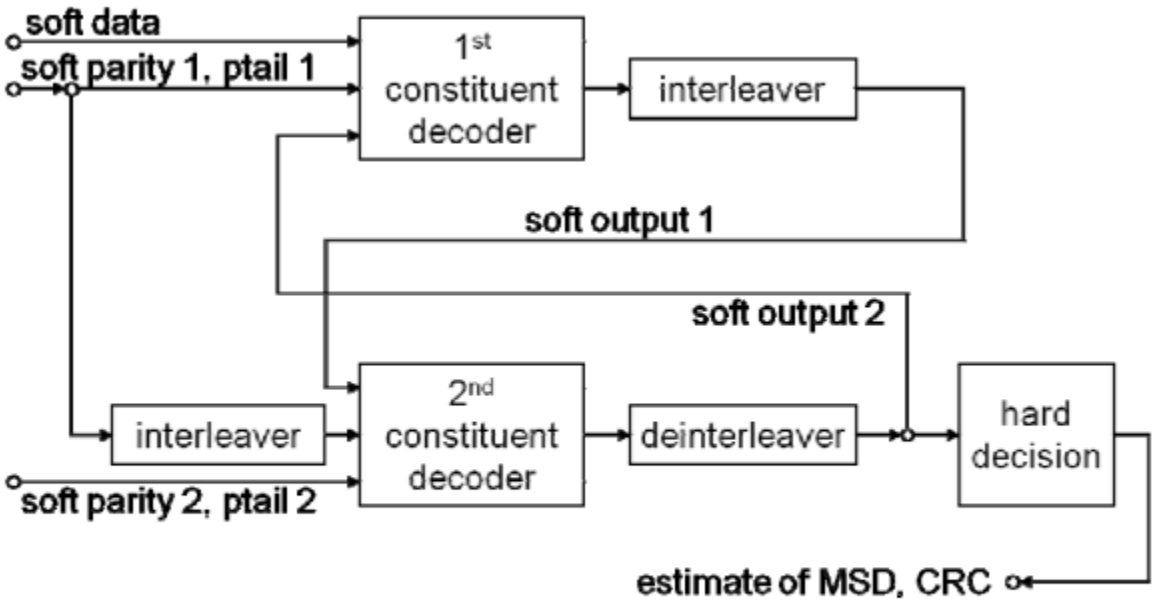


Figure 18: Turbo decoder

See 3GPP TS 26.267, at 25.

The source code associated with the figure indicates use of the BCJR algorithm:

```
/* iterative decoding */
for (i = 0; i < FEC_ITERATIONS; i++) {
  memcpy(Le12, Le21, sizeof(IntLLR)*(NRB_INFO_CRC + NRB_TAIL));

  /* add received systematic bits to extrinsic information */
  for (j = 0; j < NRB_INFO_CRC+NRB_TAIL; j++) {
```

```
    temp = (Int32)Le12[j] + (Int32)syst1[j];
    Le12[j] = (ABS(temp) < LLR_MAX) ?
      (IntLLR)temp : (IntLLR)(SIGN(temp)*LLR_MAX);
  }
  /* decode code one (produces Le12) */
  Bcjr(parity1, Le12);

  /* interleave extrinsic information (produces interleaved Le12) */
  Interleave(Le12, Le21);

  /* add received systematic bits to extrinsic information */
  for (j = 0; j < NRB_INFO_CRC; j++) {
    temp = (Int32)Le21[j] + (Int32)syst1[interleaverSeq[j]];
    Le21[j] = (ABS(temp) < LLR_MAX) ?
      (IntLLR)temp : (IntLLR)((SIGN(temp))*LLR_MAX);
  }
  for (j = 0; j < NRB_TAIL; j++) {
    Le21[j+NRB_INFO_CRC] = syst2[j];
  }
  /* decode code two (produces interleaved Le21) */
  Bcjr(parity2, Le21);

  /* deinterleave extrinsic information (produces Le21) */
  Deinterleave(Le21);
 }
```

See ecall-fec.c, lines 239-268, 3GPP TS 26.268, at 21; *see also id.*:

- ecall-fec.c line 318

```
  /* normalization of betaQ */
  for (i = 0; i < FEC_STATES; i++) {
    temp = (Int32)bTemp1[i] -  (Int32)norm;
    bTemp1[i] = (temp < (-LLR_MAX)) ? (IntLLR)(-LLR_MAX) :  (IntLLR)temp;
```

	} Data processed by the BCJR algorithm, for example, generates soft decision information: "Turbo codes are composed of an interconnection of component codes through interleavers, typically convolutional codes, and their decoders consist of an equal number of component decoders each of which operates on its corresponding codeword and shares information with other component decoders iteratively according to the topology of the encoder. The decoding algorithm in the component decoders is the maximum a-posteriori probability (MAP) algorithm typically implemented in the form known as the Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [6]. The main advantage of a MAP decoding algorithm over a maximum likelihood decoding algorithm such as the Viterbi algorithm [7] is that it produces optimum soft information which is crucial to the operation of these decoders. The BCJR algorithm was generalized in [8] into a soft-input soft-output a posteriori probability (SISO-APP) algorithm to be used as a building block for iterative decoding in code networks with generic topologies. The advantages of the SISO-APP algorithm over other forms of the MAP algorithm is that it is independent of the code type (systematic/nonsystematic, recursive/nonrecursive, trellis with multiple edges), and it generates reliability information for code symbols as well as message symbols which makes it applicable irrespective of the concatenation scheme (parallel/serial/hybrid), and hence will be considered in this paper." *See* Mansour et al., "VLSI Architectures for SISO-APP Decoders," IEEE Transactions On Very Large Scale Integration ("VLSI") Systems, Vol. 11, No. 4 (Aug. 2003), at 627, *available at* http://shanbhag.ece.illinois.edu/publications/mansr-tvlsi-2003-2.pdf. As shown by the source code associated with the figure, the soft decision information is then normalized, or "weighted," and then output to either (1) the "interleaver" memory module if performed at the first soft decision decoder or (2) the "deinterleaver" memory module if performed at the second soft decision decoder shown in the figure. Note that the variables "betaQ" and "alphaQ" comprise examples of such soft decision information in the source code: */* normalization of betaQ */* *for (i = 0; i < FEC_STATES; i++) {* *temp = (Int32)bTemp1[i] - (Int32)norm;*

	bTemp1[i] = (temp < (-LLR_MAX)) ? (IntLLR)(-LLR_MAX) : (IntLLR)temp; *}* *...* */* normalization of alphaQ */* *for (i = 0; i < FEC_STATES; i++) {* *temp = (Int32)alpha2[i] - (Int32)norm;* *alpha2[i] = (temp < (-LLR_MAX)) ? (IntLLR)(-LLR_MAX) : (IntLLR)temp;* *}* *See* ecall-fec.c, lines 318-352, 3GPP TS 26.268, at 21 (emphasis added).
performing, for a predetermined number of times, iterative decoding from the first to the last of multiple decoders, wherein an output from the last soft decision decoder is fed back as an input to the first soft decision decoder, then from the first to the second decoders, and propagate to the last decoder in a circular circuit.	The '742 Accused Instrumentalities perform, for a predetermined number of times, iterative decoding from the first to the last of multiple decoders, wherein an output from the last soft decision decoder is fed back as an input to the first soft decision decoder, then from the first to the second decoders, and propagate to the last decoder in a circular circuit. For example, the representative product, the ZenFone 7 Pro iteratively decodes, for a predetermined number of times, from the first to the last of multiple decoders, wherein an output from the last soft decision decoder is fed back as an input to the first soft decision decoder, then from the first to the second decoders, and propagate to the last decoder in a circular circuit, as shown in the figure below:

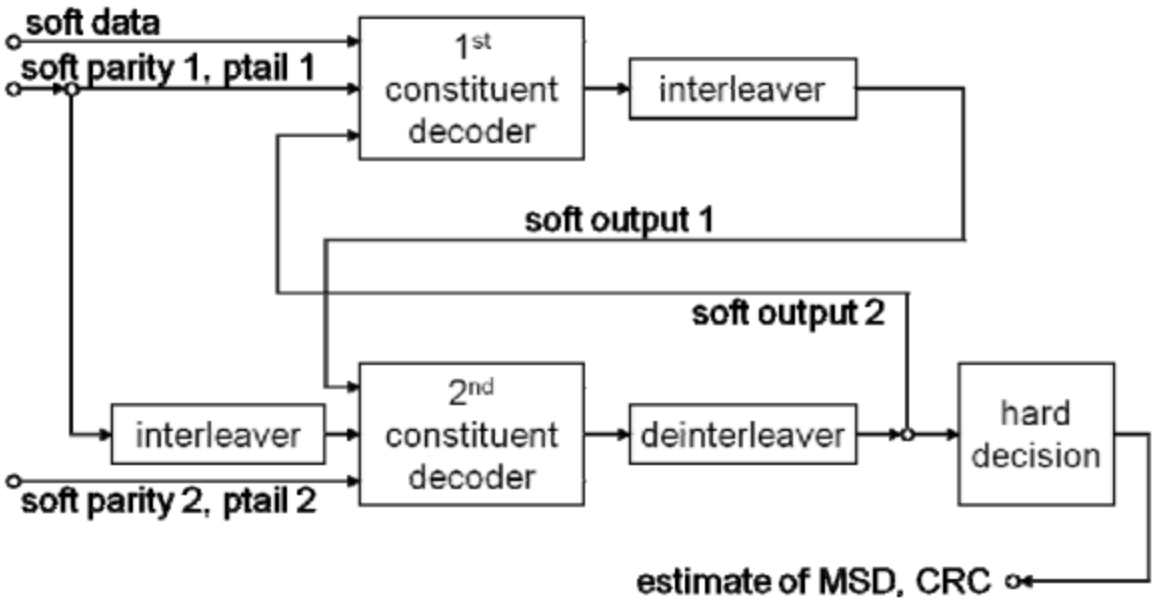


Figure 18: Turbo decoder

See 3GPP TS 26.267, at 25.

As shown in the example figure, decoding occurs from the first soft decision decoder, "1st constituent decoder," to the second, or last, soft decision decoder, "2nd constituent decoder." The second soft decision decoder outputs "soft output 2," which is fed as back as an input to the first soft decision decoder. The first soft decision decoder outputs "soft output 1." This "soft output 1" is fed as input into the second soft decision decoder. The second soft decision decoder is the last soft decision decoder in a circular circuit propagating from the first soft decision decoder to the second soft decision decoder to the first … to the second …to the first … to the second, etc.

This process is performed a predetermined number of times as defined by the software governing the turbo decoding process. For example, the default number of iterations defined in the source code associated with the figure is 8 iterations, stored in the variable FEC_ITERATIONS:

```
#define FEC_VAR                 (30206)         variance: 1/4550000 in Q37
#define FEC_MEAN                (0xB9999A)      mean: 5.8 in Q21
#define FEC_ITERATIONS          (8)             number of decoder iterations
#define FEC_STATES              (8)             number of decoder states
```

See 3GPP TS 26.268, at 10.

As another example, FEC_ITERATIONS is used during decoding process:

```
/* iterative decoding */
for (i = 0; i < FEC_ITERATIONS; i++) {
memcpy(Le12, Le21, sizeof(IntLLR)*(NRB_INFO_CRC  + NRB_TAIL));
/* add received systematic bits to extrinsic information */
        for (j = 0; j < NRB_INFO_CRC+NRB_TAIL; j++)  {
                temp = (Int32)Le12[j] + (Int32)syst1[j];
                Le12[j] = (ABS(temp) < LLR_MAX)  ?
                        (IntLLR)temp : (IntLLR)(SIGN(temp)*LLR_MAX);
        }
        /* decode code one (produces Le12) */
        Bcjr(parity1, Le12);
        /* interleave extrinsic information (produces interleaved Le12) */
        Interleave(Le12, Le21);
        /* add received systematic bits to extrinsic information  */
        for (j = 0; j < NRB_INFO_CRC; j++) {
                temp = (Int32)Le21[j] + (Int32)syst1[interleaverSeq[j]];
                Le21[j] = (ABS(temp) < LLR_MAX)  ?
                        (IntLLR)temp  : (IntLLR)((SIGN(temp))*LLR_MAX);
        }
        for (j = 0; j < NRB_TAIL; j++) {
```

	Le21[j+NRB_INFO_CRC] = syst2[j]; *}* */* decode code two (produces interleaved Le21) */* *Bcjr(parity2, Le21);* */* deinterleave extrinsic information (produces Le21) */* *Deinterleave(Le21);* *}* *See* ecall-fec.c, lines 239-268, 3GPP TS 26.268, at 21.

Caveat: The notes and/or cited excerpts utilized herein are set forth for illustrative purposes only and are not meant to be limiting in any manner. For example, the notes and/or cited excerpts, may or may not be supplemented or substituted with different excerpt(s) of the relevant reference(s), as appropriate. Further, to the extent any error(s) and/or omission(s) exist herein, all rights are reserved to correct the same.